Universidad Politécnica de Cartagena

Departamento de Tecnologías de la Información y las Comunicaciones

Desarrollo de una plataforma experimental de mecanismos de control de potencia a nivel MAC en redes de sensores

Julio J. Foulquié Vicente

Director
Javier Vales Alonso

2007

A mis padres, que me lo han dado todo
y sin los que probablemente hoy,
no sería quien soy

Autor: *Julio J. Foulquié Vicente*

Correo electrónico del Autor: *jfoulquie@gmail.com*

Director(es): *Javier Vales Alonso*

Correo electrónico del Director: *javier.vales@upct.es*

Codirector(es):

Título del PFC: *Desarrollo de una plataforma experimental de mecanismos de control de potencia a nivel MAC en redes de sensores*

Title: *Experimental testbed of MAC power control mechanism in Wireless Sensors Networks*

Etiquetas: *WSN, sensores, MAC, S-MAC, TPC, consumo*

Titulación: *Ingeniería Técnica de Telecomunicación*

Especialidad: *Telemática*

Departamento: *Tecnología de la Información y las Comunicaciones (TIC)*

Fecha de Presentación: *Septiembre - 2007*

Abstract

This project was designed trying to verify the theoretical studies published in the article *"Performance Evaluation of MAC Transmission Power Control in Wireless Sensor Networks"* [1]. This article presents several analytic methods that allow to computing the energy saving provided by the use of transmission power control (TPC) mechanisms at MAC protocols in wireless sensor networks. It concludes that, when implementing these mechanisms at a protocol such as S-MAC, energy savings could reach up to 10 %.

The goal of this project is to develop an experimental platform which allows to objectively evaluate the energy consumption in wireless sensor networks.

The project is divided in two main areas; the first one is the implementation of a *Transmission Power Control* (TPC) mechanism over *Sensor-Medium Access Control* (S-MAC) in the sensors, while the other one is the need to develop a platform which evaluates the energy savings produced by this implementation.

While tests were done under controlled conditions, some irregularities were found. This opens a new develop line which would allow to explain these irregularities and, thus, avoid them.

Resumen

El presente proyecto nace como un intento de verificación de los estudios teóricos publicados en el artículo *"Performance Evaluation of MAC Transmission Power Control in Wireless Sensor Networks"* [1]. En dicho artículo se desarrolla una serie de métodos analíticos que permiten calcular el ahorro de energía proporcionado por la implementación de mecanismos TPC en protocolos *Medium Access Control* (MAC) para redes de sensores inalámbricos. El artículo concluye que, implementando estos mecanismos en un protocolo como S-MAC el ahorro podría llegar a ser de hasta un 10 %.

El objetivo que se plantea es el desarrollo de una plataforma experimental que permita una evaluación objetiva del consumo energético en nodos de redes de sensores inalámbricos.

El proyecto se divide en dos grandes áreas, por un lado se aborda la implementación en los sensores del TPC sobre S-MAC mientras que por otro se presenta la necesidad de desarrollar una plataforma que permita evaluar el ahorro energético que produce dicha implementación.

Por otra parte, aunque la realización de las pruebas tuvo lugar en condiciones controladas, se presentan algunas irregularidades en las medidas que abren una futura línea de desarrollo que permita explicarlas y, en consecuencia, evitarlas.

Agradecimientos

A través de estas líneas quiero expresar mi agradecimiento a todas aquellas personas que han contribuido de alguna manera a la realización de este proyecto.

No cabe duda de que al final, este proyecto, es el resultado de una larga lista de contribuciones sin las cuales, probablemente, no hubiese podido terminar nunca.

Para empezar quiero agradecer a Javier Vales la tutela y apoyo prestados durante todo el proyecto así como la oportunidad de realizar mi proyecto con él.

Está claro que es imposible mencionar a todos los que han contribuido pero me gustaría agradecer especialmente a todos los profesores que, de una forma u otra me han ayudado poniendo a mi disposición sus conocimientos, ellos saben quienes son.

No me gustaría olvidar a todos los compañeros de clase que siempre me han dado ánimos, incluso en los peores momentos. ¡Muchas gracias! Sin vosotros Telemática nunca habría sido igual.

Por supuesto no podría dejar de mencionar a mis compañeros de piso, que me han acompañado en esta travesía convirtiéndose en grandes amigos y haciéndome pasar muchos de mis mejores momentos.

Finalmente agradecer a toda mi familia el simple hecho de estar ahí ya que, sin saberlo, han supuesto un gran apoyo para mí. Un agradecimiento especial merece Inma que ha colabroado activamente y, especialmente, mis padres y mi hermano.

Índice general

1. Introducción **1**
1.1. Objetivos 3
1.2. Estructura de este proyecto 4

2. Hardware y Software utilizado **5**
2.1. Los Dispositivos 5
2.1.1. Sensores 5
2.1.2. Alimentación de los sensores 6
2.1.2.1. Baterías 6
2.1.2.2. Alimentación externa 6
2.1.3. Consideraciones de la antena 6
2.1.4. Almacenamiento de datos 8
2.1.5. Sensor Boards 9
2.1.6. Conector de expansión 10
2.2. Plataforma de programación 11
2.2.1. Programación de los dispositivos 11
2.2.2. ISP 12
2.2.3. Botón de Reset 12
2.2.4. Alimentación del programador 12
2.3. Lenguajes de programación para los sensores 12
2.3.1. nesC 13
2.3.1.1. Estructura de un componente 14
2.3.1.2. Tipos de datos 16
2.3.1.3. Tipos de funciones 16
2.3.1.4. Otras funcionalidades de nesC 17
2.4. Sistemas operativos para los dispositivos 18
2.4.1. Introducción 18
2.4.2. TinyOS 20
2.4.2.1. ¿Qué es TinyOS? 20
2.4.2.2. Componentes primitivos de TinyOS 20
2.4.2.3. Estructura de TinyOS 22

3. Protocolos MAC en WSN **23**
3.1. Clasificación de protocolos MAC 24
3.2. Elección del protocolo MAC 25
3.3. Protocolo S-MAC 26
3.3.1. Introducción 26
3.3.2. Mecanismo de control de transmisión de potencia 27

4. Implementación del TPC en S-MAC **31**
4.1. Implementaciones de S-MAC 31
4.2. El S-MAC de la USC 32
4.3. Primeros pasos 33
4.4. Implementación real del TPC sobre S-MAC 36
4.5. SMACTest 42

5. Implementación del circuito de medidas de consumo **45**

6. Pruebas y resultados **49**
6.1. Medidas de consumo 50

7. Problemas encontrados **55**
7.1. Problemas software 55
7.1.1. Aplicación de prueba 55
7.1.2. Potencia de transmisión por defecto 56
7.2. Problemas hardware 57
7.2.1. Primera solución 57
7.2.2. Segunda solución 58
7.2.3. Tercera y última solución 60
7.2.4. Fuentes de alimentación 62

8. Conclusiones y líneas futuras **65**

A. Esquemas y circuitos de los MICA2 **67**

Índice de figuras

1.1. Red inalámbrica de sensores 2
1.2. Posible arquitectura de una WSN para la detección de incendios 2

2.1. Logotipos de algunos fabricantes de Hardware para WSN 5
2.2. Los modelos MICA2 y MICA2DOT de la marca Crossbow 6
2.3. Especificaciones de consumo de corriente para la serie MICA2 7
2.4. Gráfica V_{RSSI} vs. Potencia (dBm) 8
2.5. Basic Sensorboard 9
2.6. Placa de sensores para la plataforma MICA (Mica Sensorboard) 9
2.7. Conectores (macho y hembra) de expansión del MICA2 10
2.8. Pines del conector de expansión 10
2.9. Placa de programación para los sensores MIB510 11
2.10. Diagrama de bloques del funcionamiento de la placa MIB510 11
2.11. Esquema de componentes en una aplicación genérica en nesC 14
2.12. Esquema de un componente genérico 15
2.13. Ejemplo de un componente 16
2.14. Esquema de ejecución de una aplicación nesC 17
2.15. Servidor VNC ejecutándose sobre Contiki en un micro Atmel AVR. . . . 19
2.16. Compilación de una aplicación TinyOS 20

3.1. Modelo genérico por capas para WSN 27
3.2. Capas física y MAC 28
3.3. Representación temporal de la actividad en S-MAC 28
3.4. Distancia entre nodos 29

4.1. Esquema de relaciones entre componentes de S-MAC 32
4.2. Esquema de relaciones del componente PhyRadio 33
4.3. Plataforma de programación 36
4.4. Captura de Hyperterminal recibiendo datos de los sensores 37

5.1. Esquemático del PCB construido para tomar las medidas 45
5.2. Orcad Layout 46
5.3. Layout del circuito integrador 46

6.1. Primeras pruebas 50
6.2. Sensor sujeto a una silla 51
6.3. Pruebas realizadas en el exterior 51
6.4. Imágen de una de las pruebas de medida 52
6.5. Resultados de consumo para las distintas pruebas realizadas 52
6.6. Prueba con el osciloscopio 53

7.1. Imagen de un osciloscopio Yokogawa DL-1540 58
7.2. Conector Jack de 3.5 mm estéreo 58
7.3. Conector Jack 3.5 mm desmontado 59
7.4. Captura del Soundcard Oscilloscope 59
7.5. Circuito integrador realizado con un amplificador operacional 60
7.6. Circuito integrador montado en el laboratorio 61
7.7. Montaje inicial 62
7.8. Conector ATX puenteado 62
7.9. Conector ATX manipulado 63

Capítulo 1

Introducción

En los últimos años se ha producido un gran incremento de la presencia de las comunicaciones inalámbricas en la sociedad. A nadie le sorprende hoy contemplar pequeñas redes inalámbricas desde un teléfono con tecnología bluetooth con un dispositivo manos libres, o de una PDA con un PC o cualquier tecnología inalámbrica similar. Por otra parte, cada vez más, en los hogares se dispone de conexión a Internet mediante tecnologías tales como WiFi estandarizada por el IEEE (802.11x). En definitiva, no es de extrañar que cada vez más, las investigaciones y nuevos desarrollos se centren en tecnologías inalámbricas.

Con creciente frecuencia, comienzan a aparecer sistemas y servicios basados en tecnologías inalámbricas que mejoran los procedimientos que tradicionalmente, requerían una interacción directa por parte del ser humano. Uno de los más claros ejemplos de este tipo de sistemas son las redes de sensores inalámbricos (*Wireless Sensors Network* (WSN)). El uso de este tipo de redes presenta un futuro emocionante en el que la forma en la que nos relacionamos con nuestro entorno puede cambiar de forma sustancial. Este tipo de redes abre un horizonte nuevo de aplicaciones y servicios conscientes de su entorno en el que las posibilidades son infinitas.

Las redes de sensores inalámbricos están formadas por un conjunto de nodos autónomos capaces de realizar tareas tales como monitorización ambiental o industrial. Existe una amplia variedad de sensores así como una amplia variedad de situaciones para "sensorizar"; los hay de temperatura, de presión, de velocidad, de luminosidad, de aceleración, de volumen, de presencia y así hasta un sinfín de actividades que potencialmente requieran monitorización. Si además de la utilidad como sensores les añadimos la posibilidad de formación de redes ad-hoc mediante tecnología inalámbrica, se entiende fácilmente el motivo del gran auge que están teniendo estos dispositivos en la actualidad.

Una de las ventajas que tiene este tipo de redes es la de poder operar durante largos períodos de tiempo sin intervención humana. Por ejemplo, podría instalarse una red de sensores en entornos que son demasiado peligrosos para ser controlados por humanos (por ejemplo, monitorizar la actividad sísmica de un volcán) o en entornos en los que sería demasiado costoso mantener humanos (véase, por ejemplo, exploración espacial). Por otra parte podrían instalarse en un edificio como medida de seguridad (detectores de presencia) o en un invernadero para controlar la humedad. Las aplicaciones son muy diversas pero todas ellas con características en común, tecnología inalámbrica, bajo coste

y tamaños reducidos [2].

Además de sus prometedoras posibilidades, este tipo de redes presentan a los investigadores una serie de retos debido a sus especiales características: número muy elevado de dispositivos, limitadas prestaciones, trabajo en grupo en pro de una tarea común, entornos con condiciones adversas...

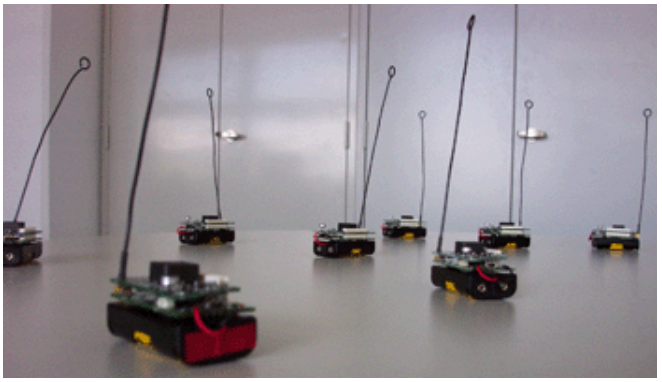


Figura 1.1: Red inalámbrica de sensores

Debido a su similitud, las redes de sensores heredan una serie de propiedades importantes de las redes ad-hoc: control descentralizado, ausencia de infraestructura de red, comunicaciones broadcast, canal de transmisión compartido, topologías de corta duración, redes multisalto... Además, cumplen una serie de condiciones propias tales como **tolerancia a fallos** (que toda la red no dependa de un nodo y que si uno cae, no afecte de sobremanera a la red), **escalabilidad** (el número de sensores en funcionamiento puede variar dependiendo del uso; dependiendo de la situación se podrían necesitar cientos, miles, e incluso millones de sensores), y un **mínimo coste de producción** (dado que hablamos de redes con un gran número de nodos, el coste por nodo debería ser bastante bajo). Una de las posibles aplicaciones de estas redes sería la detección de incendios, en este caso, la arquitectura de red sería algo similar a esto:

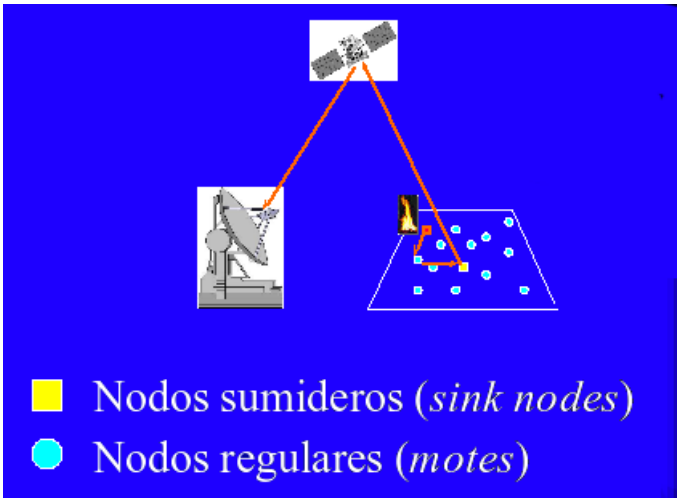


Figura 1.2: Posible arquitectura de una WSN para la detección de incendios

Por supuesto, los dispositivos para este tipo de redes deben tener un **reducido consumo de energía** (ya que los sensores son dispositivos electrónicos, van alimentados con una fuente limitada de energía -pilas convencionales generalmente- el consumo de ésta se presenta como uno de los principales problemas de las WSN). Por otra parte, tienen una serie de limitaciones importantes como pequeña capacidad de cálculo, mínimo espacio de almacenamiento y una capacidad reducida de comunicación [4]. Casi todas estas limitaciones vienen impuestas, principalmente por un motivo: **suministro limitado de energía**.

El consumo energético es uno de los factores clave de las WSN ya que, si se quieren desplegar redes del orden de los cientos o miles de nodos [3], no es factible sustituir las baterías de éstos una vez que termine su carga. Debido a esto, la vida útil de las baterías se convierte, automáticamente, en la vida útil del dispositivo. Por este motivo es conveniente desarrollar mejoras en el uso de la energía que hacen los dispositivos para permitir un ahorro energético y por consiguiente, alargar la vida útil de los aparatos. El objetivo de este proyecto es obtener un uso eficiente de los recursos del canal radio ya que está demostrado que el mayor gasto energético en los sensores se debe a la potencia gastada en la etapa radio. La transmisión de un solo bit equivale en gasto energético a la ejecución de cientos de instrucciones en el procesador. Para ello se pretende realizar una modificación del protocolo de transmisión que utilizan los sensores. En capítulos posteriores se trataran más a fondo los protocolos más usados en la actualidad.

1.1. Objetivos

El objetivo de este proyecto es desarrollar una plataforma que permita evaluar el consumo energético de los sensores, no solo de los utilizados como prueba en este proyecto si no también de cualquiera existente. De esta manera será posible evaluar el rendimiento de nuevos protocolos que nos permitan obtener un uso eficiente de los recursos del canal radio, ya que está demostrado que el mayor gasto energético en los sensores se debe a la potencia gastada en la etapa radio.

Por otra parte, a su vez para poder evaluar el consumo con una prueba real se debe realizar una implementación real para lo cual se ha elegido el uso del protocolo S-MAC, que si bien no es el más idóneo para implementar mecanismos de control de potencia [1], si es de alguna manera un protocolo muy didáctico que nos ayudará a adentrarnos en el mundo de los sensores inalámbricos, de su programación, de su pila de protocolos y de su funcionamiento a fin de cuentas. Para ello, se pretende emplear el mecanismo de Control de Potencia de Transmisión (TPC) en la que los datos se envían a la mínima potencia requerida y la secuencia RTS-CTS-ACK a la potencia nominal (máxima). Con este mecanismo, se usará el protocolo S-MAC en la capa MAC y el sistema operativo TinyOS. Los nodos sensores de la Red de Sensores Inalámbricos serán los Mica2. Para cumplir con nuestros objetivos, se tendrá que modificar la potencia a la que se transmite los paquetes de control (RTS,CTS,ACK) para que lo hagan a la máxima potencia de transmisión. Una vez se ha implementado el TPC, se harán pruebas para comprobar el ahorro energético, conociendo de antemano que, para S-MAC, el ahorro es bastante menor que en otros protocolos de la capa MAC [1]. Las pruebas se realizarán mediante el depurador implementado (ver 4

de S-MAC (a través del *hyperterminal*) y midiendo, mediante un circuito integrado y un osciloscopio, la cantidad de energía empleada en el funcionamiento de los nodos y a lo largo de un período de tiempo determinado.

En resumen, este proyecto pretende, además de la creación de un sistema que permita evaluar el consumo energético de los sensores, llevar a cabo la implementación del mecanismo de control de potencia (TPC) en el protocolo S-MAC, implementado para los sensores MICA2 de la empresa Crossbow [8]. Además, es objetivo también intentar realizar al implementación real de los cálculos expuestos en el artículo *"Performance Evaluation of MAC Transmission Power Control in Wireless Sensor Networks"*[1] para conocer si realmente merece la pena implementar el TPC en otros protocolos.

1.2. Estructura de este proyecto

Este proyecto está compuesto, en primer lugar, por un capítulo de introducción que sirve, a su vez, como resumen de los objetivos del trabajo realizado. A continuación se hace una pequeña presentación de los dispositivos utilizados así como del software usado sobre dichos dispositivos. Tras la muestra del hardware y el software usado, se presenta una perspectiva general acerca de los protocolos MAC, sus características, sus requerimientos y algunos ejemplos de ello, centrándose por supuesto en el protocolo S-MAC, que es el empleado aquí. Dentro de dicho capítulo se explican también algunos conceptos acerca del TPC, su funcionamiento y sus posibles ventajas. Tras la descripción teórica, el siguiente capítulo explica desde el principio los pasos llevados a cabo para implementarlo sobre el protocolo S-MAC. Como ya se comentó en los objetivos, se pretende desarrollar una plataforma que permita la evaluación del consumo energético de los sensores y su creación se explica en el capítulo siguiente. A continuación se desarrolla un capítulo en el que se cuenta la experiencia al hacer las pruebas y tomar los resultados y se dan algunos motivos que pretenden justificar los mismos. Acercándonos al final tenemos un capítulo que gira en torno a la problemática encontrada durante la realización del proyecto, tanto de hardware como de software y, por último, tenemos las conclusiones que se pueden extraer del proyecto así como posibles líneas futuras que podría tomar el mismo.

Capítulo 2

Hardware y Software utilizado

2.1. Los Dispositivos

Las redes de sensores inalámbricos cuentan con varios fabricantes de nodos sensores:

- **CROSSBOW:** [8] especializada en los sensores, desarrolla plataformas hardware y software que dan soluciones para las WSN. Entre sus productos, encontramos las plataformas MICA, MICA2, MICAZ, MICA2DOT, TELOS y TELOSB.
- **MOTEIV:** [9] Joseph Polastre desarrolló Tmote Sky y Tmote Invent. En la actualidad fabrican tecnologías para las WSN así como soluciones específicas usando dichas tecnologías.
- **SHOCKFISH:** [10] empresa suiza que desarrolló TinyNode.

Figura 2.1: Logotipos de algunos fabricantes de Hardware para WSN

Las pruebas han sido realizadas utilizando sensores de la marca Crossbow; en concreto, el modelo MPR410 de la serie MICA2. Esta sección describe la plataforma hardware usada y el sistema operativo que utilizan (TinyOS).

2.1.1. Sensores

Los sensores que se han elegido para realizar las pruebas efectuadas en este proyecto han sido los MPR410 de la serie MICA2. El fabricante dispone de diversos modelos con distintos tamaños y usos; en la figura 2.2 se muestran dos de los modelos más usados en la actualidad:

Figura 2.2: Los modelos MICA2 y MICA2DOT de la marca Crossbow

2.1.2. Alimentación de los sensores

2.1.2.1. Baterías

El MPR400 (916MHz), MPR410(433MHz) y MPR420(315MHz) de la serie MICA2 son la última generación de "motas" de la empresa Crossbow. Esta serie está, por defecto, alimentada con dos baterías AA de 1'5V aunque es posible utilizar cualquier combinación de baterías (AAA, C, D) siempre que proporcione una tensión similar.

2.1.2.2. Alimentación externa

Los MICA2 pueden ser alimentados externamente de diversas maneras y el fabricante especifica que cualquier fuente de alimentación que proporcione una tensión continua entre 2'7 y 3'3 voltios es válida para suministar energía a los sensores. A continuación se puede observar las especificaciones extraídas del propio datasheet de los dispositivos:

2.1.3. Consideraciones de la antena

La interfaz radio de los MICA2 es capaz de realizar operaciones en múltiples canales dentro del ancho de banda establecido. El MPR420 puede trabajar en 4 canales de la banda de 315 *Mega Hertzios* (MHz), el MPR410 puede actuar también en 4 canales pero de la banda de 433 MHz, y el MPR400 puede operar en 50 canales en una banda desde los 902 MHz hasta los 928 MHz. El fabricante especifica que, para que no exista interferencia, es necesario que los canales adyacentes estén separados un mínimo de 500 kHz [8]. Tanto la serie MPR400(MICA2) como la MPR500(MICA2DOT) utilizan una interfaz radio compatible entre sí que permite la comunicación con nodos de la otra serie.

La potencia de transmisión en los MICA2 puede ser ajustada en un gran rango de niveles. Existe un registro que controla el nivel de potencia RF dentro del propio sensor (PA_POW, dirección 0x0B) y la siguiente tabla proporciona los valores correspondientes las distintas potencias para el MPR400:

La radio de los MICA2 también proporciona una medida de la potencia de señal recibida llamada *Receive Signal Strength Indicator* (RSSI). Esta salida es medida en el canal 0 del conversor *Analógico / Digital* (A/D) y puede ser leída desde los programas que se ejecutan. Esta medida será utilizada (como se mostrará más adelante) en este proyecto para calcular la distancia entre sensores y reducir con ello la potencia de transmisión, con

SYSTEM SPECIFICATIONS

Currents		**Example Duty Cycle**
Processor		
current (full operation)	8mA	1
current sleep	8uA	99
Radio		
current in receive	8mA	0.75
current transmit	12mA	0.25
current sleep	2uA	99
Logger Memory		
write	15mA	0
read	4mA	0
sleep	2uA	100
Sensor Board		
current (full operation)	5mA	1
current sleep	5uA	99
Computed mA-hr used each Hour		
Processor		0.0879
Radio		0.0920
Logger Memory		0.0020
Sensor Board		0.0550
Total current (mA-hr) used		**0.2369**
Computed battery life Vs. battery size		
		Battery Life (months)
Battery Capacity (mA-hr)		
	250	1.45
	1000	5.78
	3000	17.35

Figura 2.3: Especificaciones de consumo de corriente para la serie MICA2

el consiguiente ahorro de energía. La conversión desde el canal 0 del *Analog to Digital Converter* (ADC) al contador RSSI en dBm, viene dada por las siguientes fórmulas:

$$Vrssi = \frac{Vbatt \cdot ADCCounts}{1024}; \tag{2.1}$$

$$RSSI(dBm) = -51{,}3 \cdot Vrssi - 49{,}2[dBm]; \tag{2.2}$$

A continuación se muestra una gráfica que relaciona la tensión inducida en el conversor A/D y la potencia de señal recibida (en dBm) por el sensor:

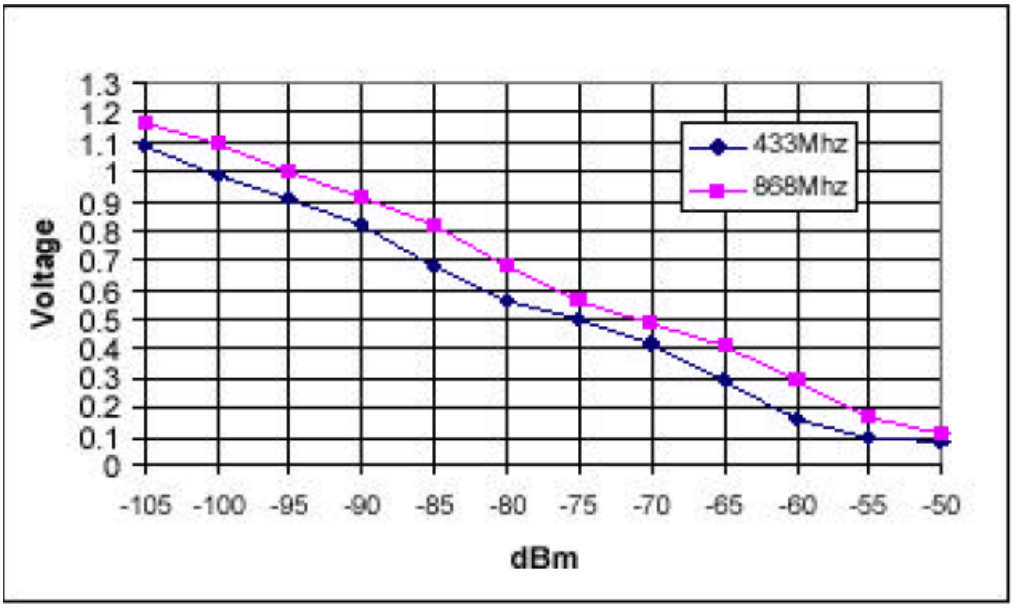


Figura 2.4: Relación entre la tensión inducida en el ADC y la potencia de señal recibida

La antena utilizada varía en función de los sensores. El alcance y la calidad de la transmisión van muy ligados al tipo de antena y a su posición dentro del entorno. Además, se debe estar seguro de que se cumple con las regulaciones internacionales en cuanto a radiación se refiere. En Crossbow, se ha usado una antena de cable de cobre sólido y considerando la longitud de la antena, existe una regla empírica: 75/frecuencia (MHz) = longitud en metros.Para los 433 MHz de un MICA2, sería necesaria una antena de cable de cobre sólido de 17.3 cm. Si se desea un incremento del rango y el tamaño es una preocupación secundaria, las antenas direccionales pueden ser atractivas.

2.1.4. Almacenamiento de datos

Las “motas“ MICA2 incluyen una memoria FLASH de 4 Mbit para almacenar datos, mediciones o cualquier tipo de información. La memoria FLASH soporta alrededor de 100.000 medidas.

2.1.5. Sensor Boards

Las series MTS de las placas de sensores y las series MDA de las placas de sensores y adquisición de datos son diseñadas para interactuar con la familia de Motes inalámbricos MICA, MICA2 y MICA2DOT. Hay una variedad de placas de sensores disponibles y son específicas al tipo del nodo sensor. A continuación se muestran dos de las placas existentes a modo de ejemplo:

- **Basic Sensorboard**
  Se trata de una placa que dispone de dos sensores (temperatura y luz). Trabaja con diversas plataformas entre las que se encuentranlos MICA. Es la placa, por defecto, para las plataformas rene y rene2.

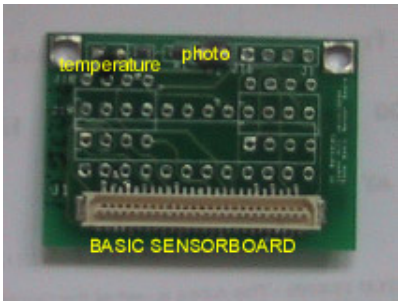


Figura 2.5: Basic Sensorboard, su uso es posible con los MICA

- **Mica Sensorboard**
  Esta placa es la utilizada por defecto para la plataforma MICA; aunque existen versiones con menos componentes que los indicados en la figura 2.6, esta es una versión bastante completa que sirve de ejemplo. Como se puede observar en la figura la placa lleva un sensor de temperatura, de luz, un magnetómetro, un acelerómetro, un micrófono y un pequeño altavoz (timbre).

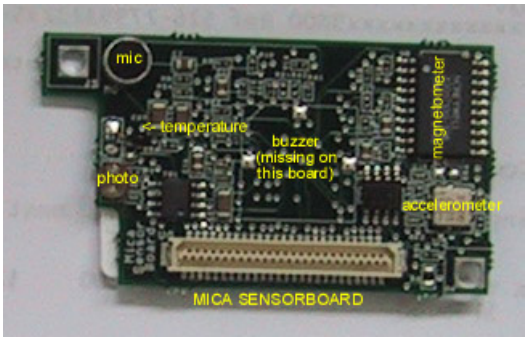


Figura 2.6: Placa de sensores para la plataforma MICA (Mica Sensorboard)

2.1.6. Conector de expansión

Los Mica2 soportan una gran variedad de sensores que pueden ser añadidos mediante su "conector de expansión". Este conector provee una interfaz que incluye, entradas de un conversor analógico/digital (ADC) para leer la salida de los sensores, alimentación y tierra, control de potencia, interfaz I2C, interfaz UART, entrada/salida digital de propósito general...

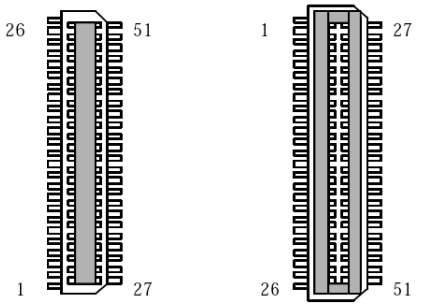


Figura 2.7: Conectores (macho y hembra) de expansión del MICA2

PIN	DESCRIPTION	PIN	DESCRIPTION
1	GND	27	UART_RXD0
2	VSNSR	28	UART_TXD0
3	INT3	29	PW0
4	INT2	30	PW1
5	INT1	31	PW2
6	INT0	32	PW3
7	BAT_MON	33	PW4
8	LED3	34	PW5
9	LED2	35	PW6
10	LED1	36	ADC7
11	RD	37	ADC6
12	WR	38	ADC5
13	ALE	39	ADC4
14	PW7	40	ADC3
15	USART1_CLK	41	ADC2
16	PROG_MOSI	42	ADC1
17	PROG_MISO	43	ADC0
18	SPI_CLK	44	THERM_PWR
19	USART1_RXD	45	THRU1
20	USART1_TXD	46	THRU2
21	I2C_CLK	47	THRU3
22	I2C_DATA	48	RESETN
23	PWM0	49	PWM1B
24	PWM1A	50	VCC
25	AC+	51	GND
26	AC-		

Figura 2.8: Relación de los pines y su función en el conector de expansión del MICA2

2.2. Plataforma de programación

El fabricante proporciona, dentro de la familia de productos MICA, unas placas de programación para los sensores que valen tanto para los MICA2 como para los MICA2DOT. La usada en este proyecto es la MIB510 que se muestra a continuación:

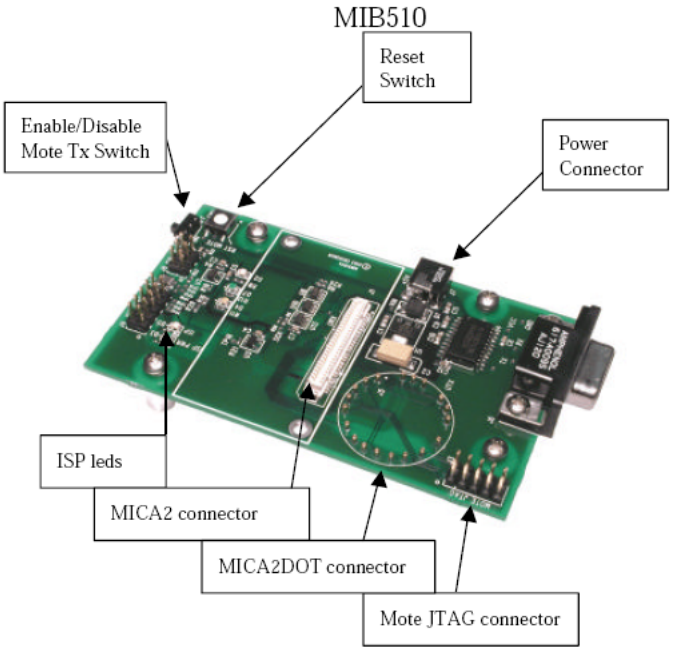


Figura 2.9: Placa de programación para los sensores MIB510

2.2.1. Programación de los dispositivos

Las placas de programación proporcionan alimentación a los sensores a través de un adaptador de corriente externo. También proporciona una interfaz serie (RS232) con un conector DB9 para la programación a través de un PC. Es importante, a la hora de programar, apagar el interruptor tanto de los MICA como de la placa, así como evitar que ésta última esté conectada a la corriente.

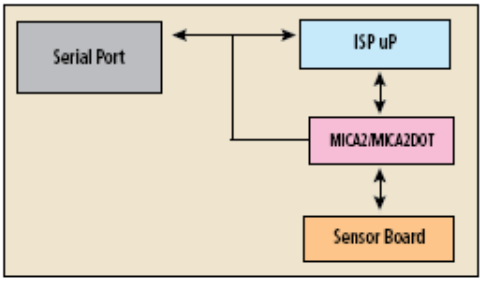


Figura 2.10: Diagrama de bloques del funcionamiento de la placa MIB510

2.2.2. ISP

La placa de programación tiene un Procesador de Sistema Integrado (*In System Processor* (ISP)) para programar los sensores. El código es descargado al ISP, a través del puerto serie, y el ISP programa el código en el sensor. El ISP y la "mota"comparten el puerto serie. El ISP trabaja a una velocidad fija de 115.000 baudios; continuamente monitoriza los paquetes que entran por el puerto serie en busca de un patrón determinado de bytes, cuando lo detecta, desactiva la interfaz serie del sensor y toma el control del puerto serie.

2.2.3. Botón de Reset

La placa dispone de un botón de reset que, al ser presionado, resetea tanto el procesador del sensor como el ISP. El pulsador resetea primero el ISP, una vez reseteado, el ISP activa el pin de reset del procesador del sensor.

2.2.4. Alimentación del programador

La MIB510 tiene un regulador de tensión que soporta entre 5V y 7V de tensión continua y proporciona a los dispositivos una tensión constante de 3V. Es importante tener cuidado a la hora de programar los MICA2 ya que la alimentación en este caso debe estar desconectada así como el interruptor de batería de los MICA, que también debe estar desconectado.

2.3. Lenguajes de programación para los sensores

La programación de sensores es complicada, entre otras dificultades está, la (como ya se comentó) limitación tanto en capacidad de cálculo como en cantidad de recursos. Por otra parte, a la hora de la creación de código no disponemos (como sí existe en los sistemas informáticos tradicionales) entornos de programación prácticos y eficientes para depurar código, simular... en estos microcontroladores todavía no hay herramientas comparables. Entre los lenguajes existentes los más utilizados en la actualidad son:

nesC: es el lenguaje por los MICA, y está directamente relacionado con TinyOS.

Protothreads: específicamente diseñado para la programación concurrente, provee hilos de dos bytes como base de funcionamiento.

SNACK: facilita el diseño de componentes para redes de sensores inalámbricas, sobre todo cuando la información o cálculo a manejar es muy voluminoso. Comparado con nesC, este lenguaje hace la programación de los dispositivos más fácil y eficiente.

galsC: diseñado para ser usado en TinyGALS, es un lenguaje programado mediante el modelo orientado a tarea, fácil de depurar, permite concurrencia y es compatible con los módulos nesC de TinyOS.

2.3.1. nesC

El lenguaje de programación **nesC** es el lenguaje empleado para la creación de aplicaciones en los MICA2. Dichas aplicaciones serán ejecutadas en nodos sensores que contengan el sistema operativo TinyOS. El hecho que hace que nesC sea un lenguaje cómodo para la programación en sensores es que está orientado a componentes. Así, se consigue una filosofía que permite abstraer al programador de detalles de bajo nivel presentes en el sistema operativo.

La idea subyacente en la orientación a componentes es que el propio sistema operativo, junto a los fabricantes de los nodos sensores, proporcione de forma intrínseca ciertos componentes ya implementados que ofrecen al programador una gran cantidad de funciones y utilidades que le permitan centrarse unicamente en la funcionalidad que desea implementar en el dispositivo, sin necesidad de tener que preocuparse por otros aspectos secundarios.

Por otro lado, nesC combina ciertos aspectos de la orientación a objetos, en el sentido de que se basa en una programación orientada a interfaces y a eventos, de manera que posee un manejador de eventos propio de este tipo de lenguajes, de forma similar a lo que ocurre en Visual Basic.

Todos los componentes que ofrece de forma intrínseca el sistema operativo se van a denominar, a partir de ahora, **componentes primitivos** y los componentes proporcionados por terceros se van a denominar **componentes complejos**.

Cuando se dice que es un lenguaje orientado a objetos, se quiere decir que todos los componentes, tanto primitivos como complejos, proporcionan unas interfaces y si un programador desea utilizar un componente, la forma de hacerlo es usando dichas interfaces, pero recordemos que son interfaces en el sentido de Orientación a Objetos, por lo que en un momento dado se podría cambiar un componente por otro siempre y cuando este otro proporcionase la misma interfaz y dicho cambio no afectase al código de la implementación de la aplicación.

De esta explicación se puede entresacar que se van a tener dos partes diferenciadas, como mínimo, en cada componente, la parte de implementación que estará programada hacia componentes y la parte de configuración que permitirá decidir que componentes son los que se utilizan para proporcionar dichas interfaces a mi componente, lo que se denomina *wiring*.

Como la forma de programar no es del todo secuencial, se recurre a la orientación a eventos, de manera que se programan las acciones que se desean realizar cuando se produzca un evento. Cada interfaz puede tener métodos y eventos y todos los componentes que implementen una interfaz deberán implementar todos sus métodos y los que las usen sus eventos.

A modo de resumen, mencionamos los principales aspectos que el modelo de programación nesC ofrece y que deben ser entendidos para el entendimiento del diseño con TinyOS:

- El modelo de nesC está formado por interfaces y componentes.
- Una interfaz puede ser usada o puede ser provista, los componentes son módulos o configuraciones.

- Una aplicación se verá representada como un conjunto de componentes, agrupados y relacionados entre sí, tal como se observa en la figura 2.11.
- Las interfaces se utilizan para operaciones que describen la interacción bidireccional; el proveedor de la interfaz debe implementar comandos, mientras que el usuario de la interfaz debe implementar eventos.
- Existen dos tipos de componentes:

  Módulos: implementan las especificaciones de un componente.

  Configuraciones: se encargan de unir o conectar (el *wiring* mencionado anteriormente) diferentes componentes en función de sus interfaces.

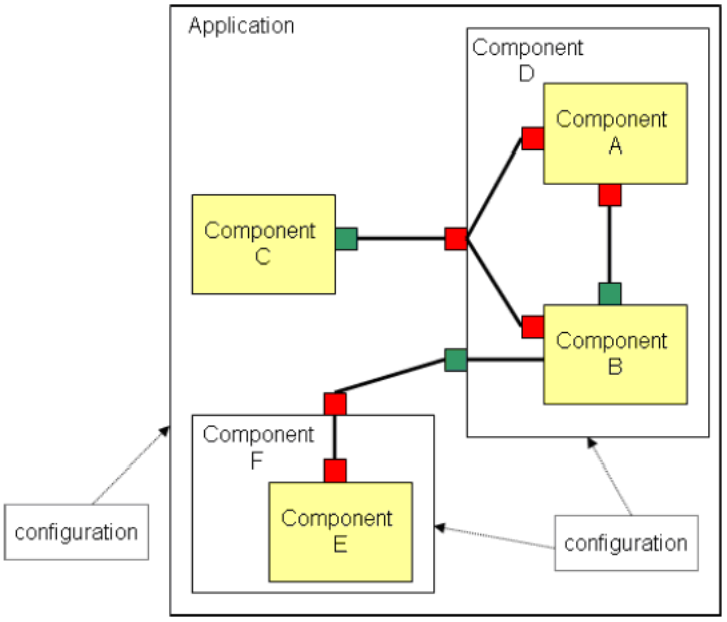


Figura 2.11: Esquema de componentes en una aplicación genérica en nesC

2.3.1.1. Estructura de un componente

Un componente está formado por tres partes: *Configuration*, *Implementation* y *Module*. Todos los componentes presentas estas partes aunque no necesariamente deben estar implementadas, es decir, puede que un componente solo tenga una implementación y una configuración pero carezca de un "módulo".

El estándar de TinyOS determina que las secciones *Configuration* e *Implementation* han de ir en un fichero que recibirá el nombre del componente con la extensión *.nc* y la sección *Module* deberá ir en otro fichero que recibirá el nombre del componente concatenado con una 'M' (M de Module), teniendo también la extensión *.nc*.

Una buena costumbre que facilita la organización y claridad del código, consiste en crear un fichero *header* o de cabecera con extensión *.h* que contenga todas las enumeraciones, registros o tipos de datos creados por el programador para la aplicación desarrollada.

La forma de ligar dicho fichero con los otros dos es utilizando al principio de los ficheros la directiva *includes header*; aunque como mención especial decir que si nos fijamos mejor en este directiva se puede ver que no se incorpora la extensión .h en la misma (al igual que ocurre en C++).

Configuración: sirve para la configuración del componente (utilizado generalmente para crear librerías). Por lo general, no va a contener nada pero tiene que estar presente.

Implementación: comúnmente, se le denomina "wiring" y no se corresponde a lo que, en una primera impresión se podría pensar que es la 'implementación'. El código de nuestra aplicación puede utilizar (*uses*) interfaces y que éstas sean proporcionadas por otro componente. Aquí es donde se especifica qué componente implementará qué interfaces.

Módulo: esta es la parte que realmente corresponde con la "implementación" del componente ya que es aquí donde de verdad se describe qué es lo que la aplicación debe hacer. Está escrito en nesC e incluye todas las tareas que deben ser ejecutadas. La estructura del módulo será algo similar a lo mostrado abajo.

```
Module PracticaM {
 Provides{ }
 Uses{ }
 Implementation{ }
}
```

Una vez descrito el componente de manera general vamos a profundizar un poco en las partes del módulo:

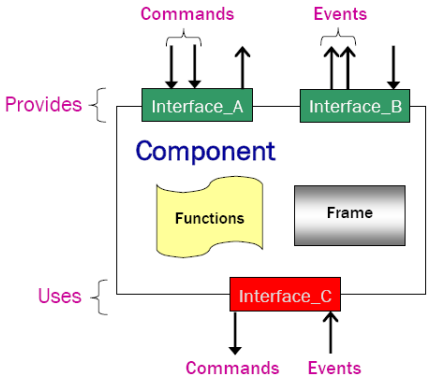


Figura 2.12: Esquema de un componente genérico

Provides: especifica las interfaces que va a proporcionar nuestro componente. Es necesario que en el código que se encuentra dentro de *implementation* estén implementados (valga la redundancia) todos los métodos que especifique la interfaz.

Uses: especifica las interfaces que utiliza nuestro componente. Estas interfaces estarán proporcionadas (*provides*) por otro componente que habrá que especificar en el fichero de conexiones (*wiring*). Cuando usemos una interfaz podremos llamar a sus métodos pero tendremos que implementar los eventos que puedan producirse.

Implementation: en esta parte es donde realmente se programa el comportamiento de nuestra aplicación. Deberá poseer las variables globales, las funciones de las interfaces que proporcione, los eventos de las interfaces que use y por supuesto, la implementación del comportamiento deseado para la aplicación.

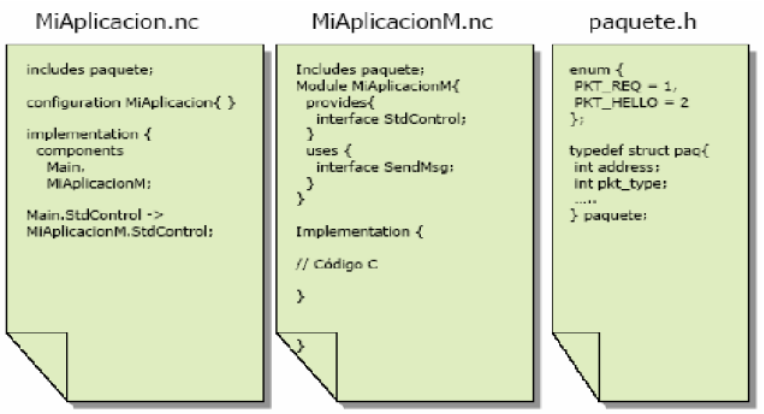


Figura 2.13: Ejemplo de un componente

2.3.1.2. Tipos de datos

Los tipos que pueden utilizarse son todos aquellos que proporciona C estándar más otros específicos de nesC, los cuales no aportan potencia de cálculo, pero son muy útiles para la construcción de paquetes ya que proporcionan al usuario información acerca del número de bits que ocupan, lo cual es muy importante a la hora de transmitir información vía radio. Los tipos adicionales son:

uint8_t: entero sin signo de 8 bits.

uint16_t: entero sin signo de 16 bits.

bool: se trata de una variable '*booleana*' que puede tomar los valores *true* o *false*.

result_t: es una variable similar a *bool* solo que puede tomar los valores predefinidos *SUCCESS* o *FAIL*. Se usa para que las funciones indiquen si su ejecución ha terminado o no con éxito.

En nesC, es posible la utilización de memoria dinámica aunque debido a su complejidad, no es muy recomendable a no ser que que sea absolutamente necesario. Para gestionar la memoria dinámica, existe un componente especial denominado *MemAlloc*.

2.3.1.3. Tipos de funciones

Debido a que nesC es un *dialecto* del lenguaje de programación C, nesC ha heredado unas serie de funciones clásicas que tienen la misma semántica que en C y con la misma manera de invocarlas. Pero existen otros tipos de funciones en nesC: *task*, *event* y *command*.

Las funciones **"command"** o comandos son funciones como las clásicas, pero que se ejecutan de forma síncrona (cuando se llaman, se ejecutan inmediatamente). La forma de llamar estas funciones es: *call interfaz.nombreFuncion*

Las funciones **"task"** o tareas se ejecutan concurrentemente en la aplicación, al igual que ocurre con los hilos o threads. Se invoca con *post interfaz.nombreTarea* e inmediatamente, tras su invocación, continúa la ejecución del programa invocador.

Las funciones **"event"** o eventos son llamadas cuando se levanta una señal en el sistema. Están basadas en la filosofía de la programación orientada a eventos, de manera que cuando el componente recibe un evento, se realizará la invocación de dicha función. Cuando un componente quiere lanzar un evento en algún momento del código debe realizar una llamada similar a *signal interfaz.nombreEvento*, además en el caso de tratarse de interfaces parametrizadas es posible utilizar la variante *signal interfaz.nombreEvento[parámetro]*.

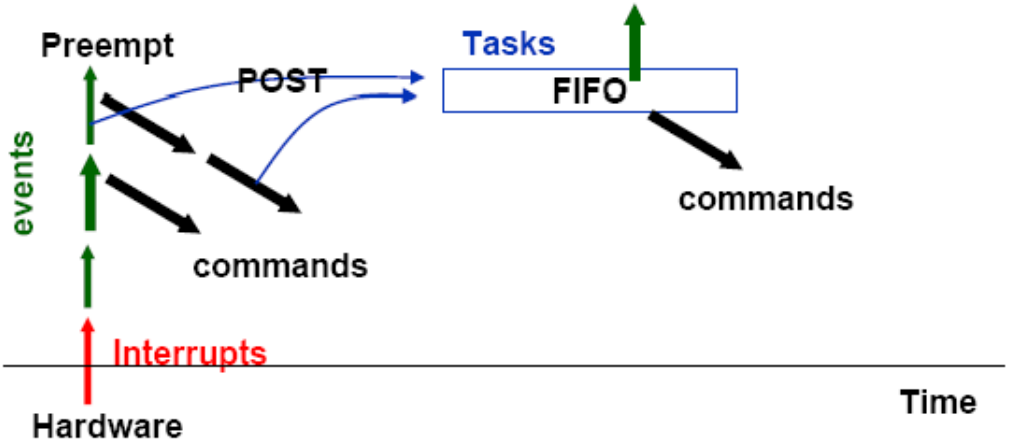


Figura 2.14: Esquema de ejecución de una aplicación nesC

Existen distintos tipos de funciones además de las nombradas, las más destacables son las funciones asíncronas; estas funciones no tienen por que realizarse de forma inmediata y, por tanto, cuando usan una variable global, se ha de indicar de forma explícita para poder realizar una exclusión mutua de la memoria y no perder ningún valor. Este tipo de ejecución, por lo general, viene dado cuando la ejecución de uno de estos tipos de función viene determinada por el levantamiento de una señal hardware, por ejemplo, que la temperatura ya se encuentre preparada para ser leída. La forma de indicarlo es anteponiendo la palabra reservada *"norace"* delante de la declaración de la variable.

2.3.1.4. Otras funcionalidades de nesC

Debido a que se permite cierto tipo de programación recurrente mediante la invocación de tareas o *task*, el lenguaje de programación permite construir una agrupación de sentencias que aseguren que, mientras que se este realizando dicha transacción, las variables implicadas no van a ser modificadas por otro hilo de ejecución. Esto se consigue mediante la palabra reservada *atomic* y la forma de usarla es similar a la mostrada en la figura 2.3.1.4.

```
atomic{
.       //...
.       //sentencias;
.       //...
}
```

Debido a que ciertas interfaces parametrizadas no pueden ser llamadas dos veces con el mismo parámetro, existe una función *built-in* que proporciona un valor único para una constante, es decir, al llamar a la función muchas veces con el mismo parámetro, nos aseguramos que cada vez va a devolver un valor distinto. La función que permite esto es:

uint_16 unique(string parámetro);

2.4. Sistemas operativos para los dispositivos

2.4.1. Introducción

Existe en la actualidad una gran variedad de sistemas operativos diseñados específicamente para trabajar con sensores inalámbricos. A continuación se presentan algunos de ellos:

Bertha: se trata de una plataforma de software diseñada e implementada para modelar, testear y desplegar una red de sensores distribuida de muchos nodos idénticos. Sus principales funciones se dividen en los siguientes subsistemas:

- Administración de procesos.
- Manejo las estructuras de datos.
- Organización de los vecinos.
- Interfaz de Red

Nut/OS: Es un pequeño sistema operativo para aplicaciones en tiempo real y CPU's de 8 bits con recursos muy limitados. Entre otras cuenta con las siguientes funciones:

- Multihilo.
- Mecanismos de sincronización.
- Administración de memoria dinámica.
- Puertos serie de Entrada/Salida.

Contiki: es un pequeño sistema operativo de libre distribución desarrollado para su uso sobre plataformas de recursos limitados, desde procesadores de 8 bits hasta sensores inalámbricos pasando por sistemas integrados sobre microcontroladores. Integra soporte para multitarea, una implementación de la pila TCP/IP e incluso es posible proporcionar una pequeña interfaz gráfica; un ejemplo de ello lo podemos ver en la figura 2.15 [7].

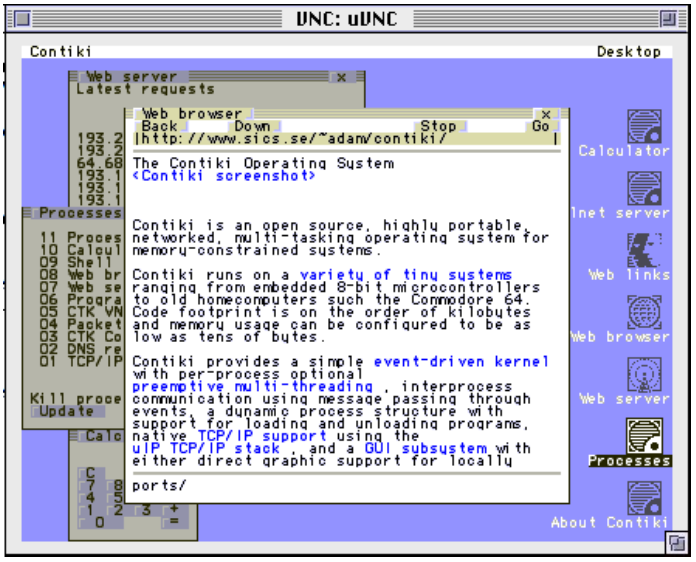


Figura 2.15: Servidor VNC ejecutándose sobre Contiki en un micro Atmel AVR.

CORMOS: (Communication Oriented Runtime System for Sensor Networks): es como su nombre indica, un sistema orientado a redes de sensores. Es un sistema muy modular, soporta concurrencia y está pensado para las limitaciones presentes en los sensores inalámbricos. Actualmente funciona con los sensores de Crossbow con micros ATMEL y Chipcon.

eCos: (embedded Configurable operating system): es un sistema operativo gratuito, diseñado para aplicaciones en tiempo real y sistemas embebidos que sólo necesitan un proceso. Es muy configurable y personalizable siendo capaz de cumplir cualquier requisito, ofreciendo la mejor ejecución en tiempo real y minimizando las necesidades de hardware.

MagnetOS: es un sistema operativo distribuido para redes de sensores, flexible y configurable que facilita el desarrollo de aplicaciones de red en dispositivos inalámbricos.

t-Kernel: es un sistema operativo que acepta las aplicaciones como imágenes de ejecutables en instrucciones básicas. Por ello, no importará si está escrito en C++ o lenguaje ensamblador.

LiteOS: este sistema operativo desarrollado en un principio para calculadoras, está siendo actualmente utilizado también para redes de sensores.

TinyOS: este es el sistema que utilizan los MICA2 utilizados en el proyecto y se describirá con más detalle en la siguiente sección.

2.4.2. TinyOS

2.4.2.1. ¿Qué es TinyOS?

TinyOS es un sistema operativo open source para redes de sensores inalámbricas y que se basa en componentes. Este sistema, junto a sus librerías y aplicaciones, está escrito en el lenguaje de programación nesC como un conjunto de tareas y procesos que colaboran entre sí y es útil para pequeños dispositivos, tales como las motas. Es un sistema operativo *"event-driven"*, lo que quiere decir que funciona a partir de eventos producidos que llamarán a funciones. Está diseñado para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en las redes de sensores. El entorno de desarrollo de TinyOS soporta directamente la programación de diferentes microprocesadores y permite programar cada tipo con un único identificador para diferenciarlo, o lo que es lo mismo se puede compilar en diferentes plataformas cambiando el atributo. TinyOS está desarrollado por un consorcio liderado por la Universidad de California en Berkley en cooperación con Intel Research.

En nesC, los programas están compuestos por componentes que se enlazan para formar un programa completo. Los componentes se enlazan a través de sus interfaces. Estas interfaces son bidireccionales y especifican un conjunto de funciones que están implementadas bien por los proveedores o bien por los que la utilizan. nesC esperará que el código que va a ser generado cree un programa con un ejecutable que contenga todos los elementos del mismo, así como los manejadores de las interrupciones de programas de más alto nivel.

En la figura 2.16, se muestra un diagrama de bloques en el que se describe el proceso de compilación para una aplicación en TinyOS.

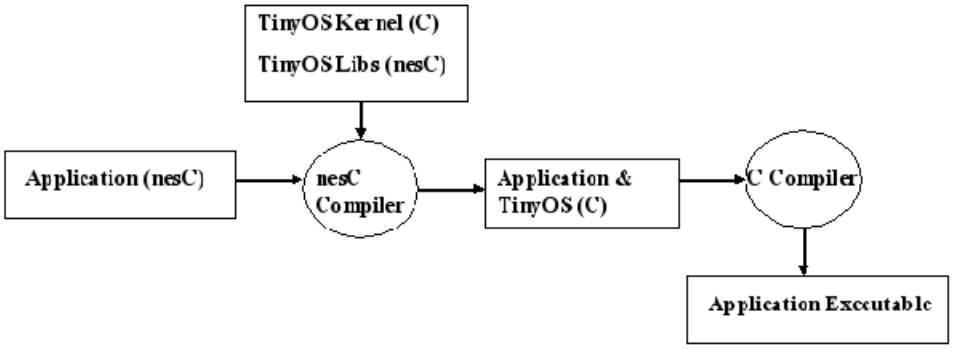


Figura 2.16: Compilación de una aplicación TinyOS

TinyOS es un proyecto de la Universidad de Berkeley; en la página de dicha universidad dedicada al proyecto es posible encontrar diversos recursos que van desde manuales de instalación hasta los repositorios con las últimas versiones del código fuente.

2.4.2.2. Componentes primitivos de TinyOS

Los tipos de componentes que podemos encontrar en TinyOS pueden ser primitivos o compuestos (proporcionados por una librería o una aplicación). Los principales compo-

nentes primitivos son:

Componente Main: representa el cuerpo main al estilo de C y necesita de la interfaz *StdControl* para su funcionamiento. Dicha interfaz será proporcionada (*provides*) por el componente que quiera convertirse en una aplicación.

Interfaz StdControl: es una interfaz especial que deben proporcionar todos los componentes que pretendan "ser una aplicación" y, por tanto, sean ejecutables en un sensor. Dicha interfaz obliga a implementar los siguientes métodos:

command result_t init();

Se ejecuta cuando el sensor arranca. Aquí es donde se inicializan las variables globales y se llama a los métodos init() de los componentes que se vayan a utilizar (no todos usados (*uses*), sino los que sean necesarios).

command result_t start();

Se ejecuta después del método *init()* y cuando la mota pasa de off a on. Es aquí donde se arrancan los temporizadores, se llama a los métodos *start()* de los componentes utilizados (sólo los necesarios)...

command result_t stop();

Se ejecuta cuando el sensor se apaga o se suspenda (estado *idle*). En esta parte del código se realizan las llamadas a los métodos *stop()* de los componentes arrancados (mediante sus métodos *start()*).

Componentes GenericComm y GenericPromiscuousComm: estos componentes proporcionan mecanismos que permiten enviar y recibir paquetes a través de la radio y de la interfaz serie. Actúa como conmutador (*switch*) entre la radio y el puerto serie de manera que, si un paquete se envía vía radio a la dirección del puerto serie, cuando lo reciba este componente, será enviado dicho paquete por el puerto serie. Utiliza el modelo AM estándar de TinyOS por lo que se basa en el envío de paquetes TOSMsg, cuya estructura está definida en */tos/types/AM.h*

Por otra parte, este componente provee (*provides*), entre otras las siguientes interfaces de interés:

SendMsg[TIPO_PAQUETE :] esta interfaz posibilita el envío paquetes a un destino concreto, estos paquetes han de ser del tipo que se especifique en el parámetro de la interfaz.

ReceiveMsg[TIPO_PAQUETE :] Esta interfaz permite la recepción de paquetes de un tipo determinado (especificado en el parámetro de la interfaz). Un sensor recibirá un paquete siempre y cuando el paquete sea enviado por un sensor perteneciente al mismo grupo, que el tipo de paquete coincida con el parámetro de la interfaz y que la dirección destino del paquete, o bien sea de broadcast,

o bien coincida con la dirección local del sensor (en el último caso, si se trata del componente GenericCommPromisuos, se recibirá siempre aunque no sea broadcast, ni vaya dirigido hacia el propio sensor).

Componente ADCC: este componente tiene la misión de ofrecer mecanismos que permitan la recolección de la información de los sensores que posea la placa. Para ello, proporciona dos interfaces:

ADC[puerto :] esta interfaz permite la conversión analógico/digital de un puerto determinado. Cada uno de los sensores conectados a la placa tiene asignado un número de puerto de manera que, mediante la especificación del parámetro de la interfaz se decide cuál de los sensores es el que se quiere procesar para obtener su valor.

ADCControl[puerto :] esta interfaz posibilita la creación de configuraciones acerca del muestreo de las muestras capturadas por el componente, de manera que se puede configurar su velocidad de muestreo o realizar un *remaping* de los puertos, para establecer una correspondencia entre los puertos software (comunes a todos los modelos de sensores) y los puertos hardware dependientes del tipo de placa sobre el que se vaya a ejecutar la aplicación.

2.4.2.3. Estructura de TinyOS

El sistema operativo tiene una estructura de directorios tal que así:

/tos/interfaces: contiene todas las interfaces que son proporcionadas por los componentes primitivos y por las aplicaciones de ejemplo.

/tos/lib: contiene librerías para resolver determinados problemas.

/tos/system: contiene todos los componentes primitivos que proporciona TinyOS.

/tos/types: contiene los tipos primitivos utilizados por los componentes de TinyOS.

/tos/platform: contiene los ficheros necesarios para la ejecución en las diversas plataformas soportadas.

/tos/sensorboard: contiene los ficheros que son específicos de cada placa.

Los esquemas y circuitos relativos a los MICA2 extraídos del manual de usuario se encuentran en el apéndice A al final del capítulo.

Capítulo 3

Protocolos MAC en WSN

Un protocolo MAC tiene que servir de base para protocolos de más alto nivel. En el caso de redes de sensores, se tiene el protocolo de enrutamiento, que usará las funciones implementadas en la MAC para enviar y recibir paquetes, sincronizar sus operaciones, etc. Las principales características que debería cumplir todo buen protocolo de acceso al medio son:

- La **flexibilidad**, porque el entorno inalámbrico es totalmente cambiante debido a interferencias en el aire de otras ondas, propiedades y formas de los materiales del entorno, y un largo etcétera. Además, los nodos pueden fallar en cualquier momento, teniendo que buscar nuevos caminos, reconfigurando la red y recalibrando los parámetros. El tráfico también puede incrementarse, ya que la información requerida en cada momento es cambiante.

- La **eficiencia**; un protocolo de MAC debe ser eficiente para poder trabajar en tiempo real, debe ser fiable y robusto ante las interferencias, tolerante a los ruidos... Además debe estar profundamente integrado con el medio donde va a trabajar, a su vez que debe ser un software "fácil de implementar" y con funciones que cubran las necesidades más amplias del mercado. Estos protocolos afectan directamente al consumo energético, ya que son la capa más próxima al nivel físico, también determinan, en parte, el coste del sistema. Así como son clave a la hora de especificar la latencia y el nivel de seguridad del sistema. En las redes de sensores, estos protocolos determinan los canales de radio a utilizar, implementan las transmisiones y recepciones a bajo nivel, además de controlar los errores. Las funciones de un protocolo MAC son, entre otras, controlar el acceso al medio compartido, que en este caso será un canal de radio (a través del aire). El protocolo debe evitar las interferencias entre transmisiones, mitigando el efecto de las colisiones, evitándolas en la medida de lo posible o detectándolas en el caso de que se den.

Los protocolos de control de acceso al medio (MAC) han sido investigados durante décadas; los diseños varían mucho según el objetivo de la aplicación. Pueden ser clasificados en categorías basadas en diferentes principios; algunos son centralizados, con una estación central como líder del grupo haciendo el control de acceso, otros son distribuidos.

Algunos usan un único canal, otros varios. Algunos usan diferentes versiones de acceso aleatorio, otros usan reserva de canal y planificación. Los protocolos están optimizados para diferentes objetivos: energía, retardo, tasa de transferencia, equidad, calidad de servicio (QoS) o soporte de múltiples servicios. Cada tipo de red necesita un protocolo diferente. Por ejemplo, las redes donde los eventos se producen de forma periódica necesitan protocolos que usen reserva y planificación del tiempo. Tendrán una mejor utilización del canal, y tendrán un mayor tiempo de vida. Por el contrario, para las redes de sensores con eventos asíncronos, el protocolo MAC ha de ser distribuido y optimizado para el ahorro de la energía. Un método distribuido usando múltiples canales y acceso aleatorio es lo más indicado para estas redes, se evita el tener un único punto de fallo, múltiples canales reducen las colisiones y retransmisiones, además del incremento de la tasa de transferencia. El acceso aleatorio evita al nodo conocer la red, lo que hace que no sea necesaria la sincronización.

3.1. Clasificación de protocolos MAC

Existen una gran variedad de protocolos MAC que fueron desarrollados con anterioridad para otros tipos de redes y que ahora han sido implementados sobre redes inalámbricas de sensores. Además, diversos grupos de trabajo han creado otros protocolos *ad hoc* para este tipo específico de redes, lo que nos da un amplia gama de protocolos; a continuación se enumeran algunos de los más significativos [17]:

CSMA *(Carrier Sense Multiple Access) Protocolos de Acceso Múltiple por Detección de Portadora* o ranurados:

- Sensor MAC (S-MAC)
- 802.15.4
- Timeout MAC (T-MAC)
- Berkeley-MAC (B-MAC)
- Power-Aware Multi-access protocol with signaling (PAMAS)

TDMA *(Time Division Multiplex Access) Protocolos de Acceso multiplexado por división de tiempo*:

- Traffic-Adaptive MAC (TRAMA)
- Low-Energy Adaptive Clustering Heirarchy(LEACH)
- Power Aware Clustered TDMA (PACT)
- Bit-Map Assisted (BMA)
- Proposed Gateway MAC (G-MAC)
- SPRIME (SupSlot Period Reservation & Inter-Master Estimation)
- SMACS (Self-Organized MAC for Sensor networks)

Otros protocolos :

- DMAC (Dynamic Topology MAC)
- CMAC (Spatial Correlation-based Collaborative)
- DSMAC (Dynamic Duty Cycle for WSN)
- SmartNode (send with same power as received)
- STEM (Sparse Topology and Energy Management)
- DPSM (Dynamic Power Saving Mechanism)
- MACA (MultiAccess Collision Avoidance)
- ZMAC (Hybrid MAC for WSN)

3.2. Elección del protocolo MAC

Como ya se ha comentado, el protocolo MAC es una de las bases tanto para la fiabilidad y velocidad de la comunicación como del consumo de los aparatos, por tanto, es muy importante a la hora de desarrollar una aplicación elegir bien el protocolo a utilizar pues, dependiendo de los requerimientos la elección será más o menos difícil. A continuación se exponen algunas de las principales características más destacadas a la hora de elegir [16]:

- La **escalabilidad**, ya que las redes de sensores son, por definición, dinámicas y la adición de nodos es algo esperable; el protocolo MAC debe estar preparado para cambios en el número de nodos y debe soportarlos sin dificultad.
- Debe ser posible el predecir **los retrasos**, el protocolo debe implementar mecanismos que eviten a la aplicación preocuparse de cosas como la disposición de los nodos, la proximidad entre ellos, la calidad del canal...
- La **adaptabilidad** a los cambios; al igual que es posible que el número de nodos varíe también pueden cambiar otra serie de variables (posición de los nodos, ruido en las transmisiones...) que no deben hacer que la comunicación se pierda.
- La eficiencia en la **gestión de la energía**, como principal desafío de las redes de sensores. La cantidad de energía utilizada en el envío y recepción de paquetes en las redes inalámbricas es esencial, ya que, como ya se vió en el primer capítulo, a menor energía utilizada, mayor tiempo de vida para la red.
- La **fiabilidad**; los protocolos MAC deben (en la medida de lo posible) evitar los bloqueos, la pérdida de paquetes, la desaparición de nodos y, además, dar respuesta a interferencias o ataques externos a la red.

En relación al penúltimo punto (*gestión de la energía*), varios protocolos MAC han sido desarrollados para enfrentarse al agotamiento de las baterías. S-MAC, por ejemplo, divide la estructura de envío, recepción de mensaje en períodos de escucha y sueño (durante el sueño el sensor tiene un consumo mínimo) [15]. El período de escucha esta dividido

en un período de sincronización y otro de transferencia de datos. La sincronización permite a los nodos anunciar periódicamente su planificación de tiempos, corrigiendo así los desplazamientos temporales propios de un sistema impreciso. Además, permite a la red sincronizar los períodos de sueño de los nodos, formando así un conjunto virtual de nodos que activan al mismo tiempo los períodos de escucha y sueño. El *Time-Medium Access Control* (T-MAC) es similar al S-MAC, pero con un período de escucha adaptativo basado en el tráfico de la red. Con T-MAC, los nodos pueden ir directamente a la fase de sueño tan pronto como el tráfico de la red se termine[1]. A continuación, se detallan los cinco puntos clave para desarrollar un protocolo MAC que presente un consumo mínimo en una red de sensores inalámbrica:

1. Las colisiones deben ser evitadas siempre que sea posible, ya que la retransmisión produce un innecesario consumo de energía y además posibles retrasos asociados. Por otro lado prevenir y evitar las colisiones puede producir una sobrecarga sustancial en la red, lo que deriva en un consumo mayor energía. La solución intermedia es uno de los factores clave.

2. A su vez, la transmisión de sobrecarga en el protocolo debe ser reducida tanto como sea posible, lo que incluye los paquetes dedicados al control de la red y los bits de cabecera de los paquetes de datos.

3. En los sistemas inalámbricos típicos, la radio del receptor ha de estar encendida siempre dando como resultado un aumento del consumo de energía. De nuevo, esto es más importante en una radio de corto alcance que en un sistema inalámbrico típico, como una red de computadores o telefonía móvil, ya que un gran porcentaje de la energía consumida ocurre cuando la radio está encendida. La situación ideal sería que la radio se enciendese solo cuando el nodo necesite enviar o recibir paquetes sin que ello conlleve un aumento en los esfuerzos de monitorización.

4. Hay puntos fundamentales en el diseño de sistemas inalámbricos entre estos parámetros: eficiencia del uso de ancho de banda, retraso, calidad del canal, consumo de energía...

5. El último, pero no menos importante principio es la capacidad de adaptación y movilidad. Para un sistema de gran escala con movilidad limitada, sería bueno un algoritmo adaptativo que funcionase con el mínimo sobrecoste posible.

3.3. Protocolo S-MAC

3.3.1. Introducción

S-MAC es un protocolo de acceso al medio que, siguiendo el paradigma de la estructuración por capas, se encuentra entre la capa física y la capa de enrutamiento [14], además es el primer protocolo de control de acceso al medio (MAC) diseñado para redes de sensores inalámbricos (WSN). Como el limitado consumo de energía es una constante en este tipo de redes, la idea básica en su creación fue la necesidad de ahorrar energía.

Por este motivo, durante el funcionamiento normal del sensor existe una alternancia entre períodos de actividad y períodos de sueño.

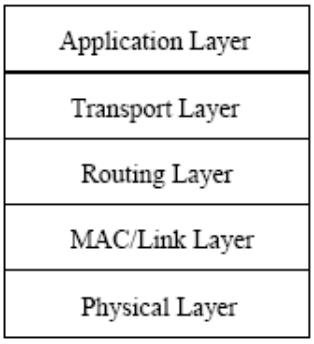


Figura 3.1: Modelo genérico por capas para WSN

Ampliando un poco la información de las capas física y MAC, se pueden obtener más detalles de las mismas como muestra la figura 3.2.

Cuando existe una baja tasa de envío y una alta latencia, el protocolo es eficiente energéticamente, ya que los receptores permanecen 'durmiendo' durante mucho tiempo.

La sincronización es necesaria en un protocolo donde los nodos sensores dependen unos de otros para establecer sus tiempos actividad y 'sueño' en la radio. Para conseguir dicha sincronización, existe un intercambio, entre los vecinos, de mensajes de sincronización llamados SYNC.

Otro problema a cubrir es el del "terminal escondido". Para solucionar este conocido problema el protocolo se emplea la secuencia de paquetes RTS-CTS (*Request to Send - Clear to Send*). A través del paso de mensajes, un nodo podrá transmitir un conjunto de paquetes usando una única secuencia RTS-CTS para negociar. De esta manera, un paquete que reciba el mensaje *Request To Send* (RTS), sabrá la longitud del paquete que va a recibir y, por tanto, el momento en el que podrá dormir.

3.3.2. Mecanismo de control de transmisión de potencia

El mecanismo de control de potencia de transmisión (TPC) en la capa MAC de las redes de sensores inalámbricos es un método para ahorrar energía. Los paquetes de datos serán transmitidos con la potencia mínima requerida y los demás paquetes, de control, serán transmitidos a la potencia nominal (máxima potencia). Este mecanismo puede ser implementado en el hardware de los actuales sensores y puede ser directamente aplicado a muchos protocolos MAC ya creados para las WSN.

El mayor gasto energético en los nodos se da en la comunicación vía radio. El consumo de la interfaz radio depende de su estado, el cual puede ser uno de los siguientes:

- **Transmission state:** transmisión del paquete donde el consumo de potencia es proporcional a la potencia de radio transmitida y puede ser seleccionada de entre un conjunto de valores discretos (en caso de las motas Mica2, de -20 a 5 dBm [8]). Sin embargo, en la práctica la potencia de salida se mantiene en un alto valor fijo.

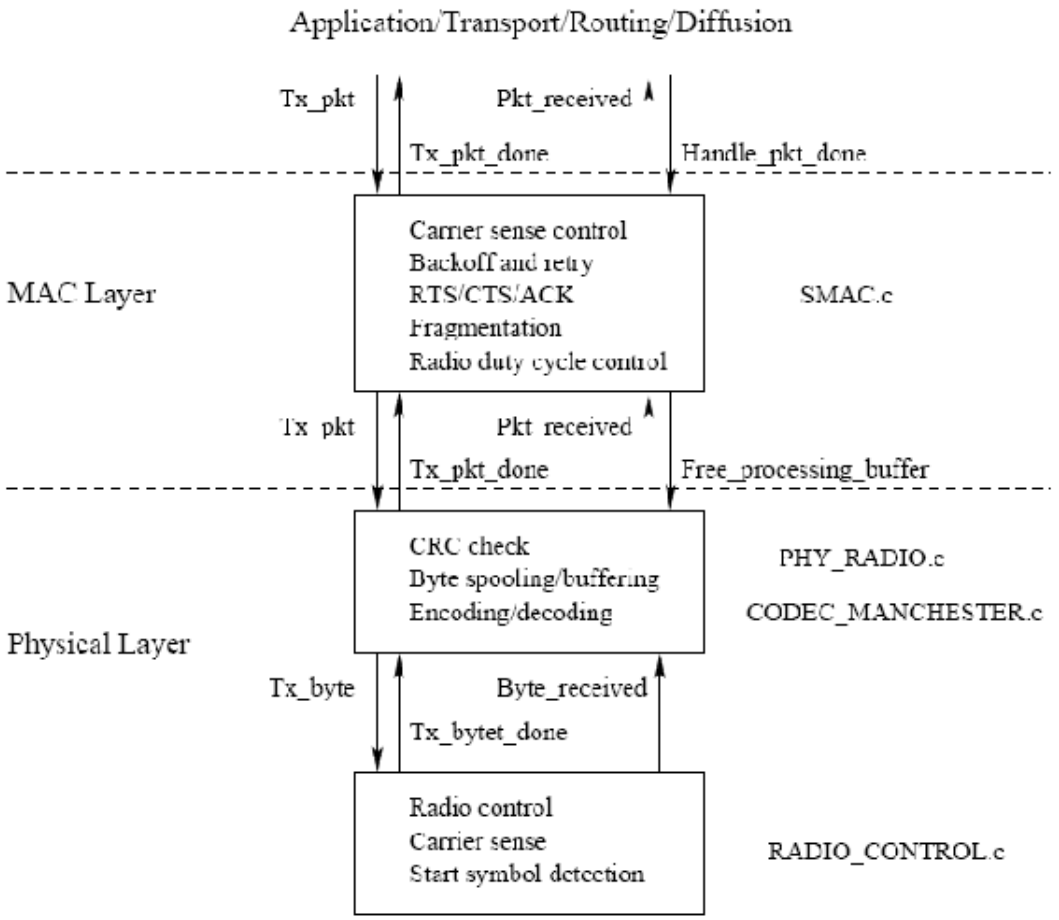


Figura 3.2: Capas física y MAC

- **Reception -and listening- state:** recepción del paquete y escucha del canal (detección de portadora). Se consume una significante suma de potencia tanto al recibir los datos como en la escucha (en las motas Mica2, son 35.4 mW [8]).
- **Sleep state:** la radio está apagada. Existe un bajo consumo de potencia debido a la alimentación del microcontrolador y otros componentes (alrededor de 3 μW en los Mica2 [8]).

Desde la perspectiva de la capa MAC, el consumo puede ser reducido si los nodos duermen durante los períodos de inactividad en lugar de mantenerse en el estado de recepción. Por tanto, el consumo medio será significativamente menor. Dicha estrategia requiere la sincronización entre los nodos (todos ellos deben dormir y despertarse al mismo

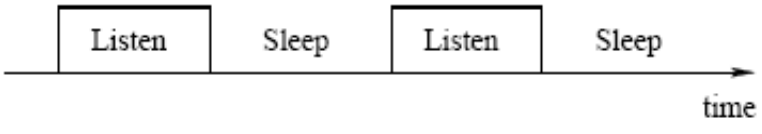


Figura 3.3: Representación temporal de la actividad en S-MAC

tiempo) así como la comunicación entre ellos. El método usado en este proyecto consiste en la reducción de la potencia de transmisión sólo durante el envío de datos (manteniendo la potencia nominal para los paquetes de control) [1].

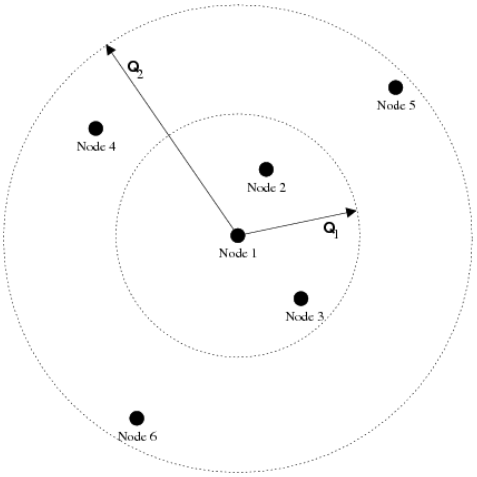


Figura 3.4: Dependiendo de la distancia entre los nodos se ajusta la potencia de transmisión

En la figura 3.4, el nodo 1 alcanza a los nodos 2 y 3 con la potencia Q1, sin embargo los nodos 4, 5 y 6 sólo podrán ser alcanzados con la potencia Q2. En este caso, Q2 es la potencia nominal que garantiza la conectividad completa. Pero si la potencia es fija, existirá un gasto de potencia innecesario cuando el nodo 1 envíe datos a los nodos 2 y 3. En este proyecto, se ha modificado la potencia de salida, por defecto, del protocolo S-MAC para que los paquetes de control (secuencia RTS-CTS-ACK) sean transmitidos a la potencia nominal (máxima en nuestro caso). Y con la implementación del mecanismo de Control de Potencia Transmitida (TPC), los datos serán enviados a la mínima potencia estimada necesaria para que lleguen a su destino.

La meta de este mecanismo es seleccionar el nivel de potencia de transmisión óptimo para no malgastar energía en el envío de los paquetes. Dicho nivel dependerá del alcance y del objetivo. Los protocolos TPC han sido comúnmente diseñados para las redes móviles ad-hoc (MANET's) con el propósito de incrementar la capacidad en medios inalámbricos por medio de la reutilización del canal o para asegurar la conectividad de la red. Dos tipos de estrategias ya han sido consideradas:

1. En la capa de red, TPC es usado para elegir el mejor subconjunto de los actuales vecinos para ser alcanzados.

2. En la capa MAC, la potencia es seleccionada para mejorar la reutilización del canal para cada paquete o para reducir la probabilidad de colisión de los paquetes.

Las últimas propuestas están basadas en mecanismos muticanal, usando los canales adicionales para señalizar las transmisiones entrantes y calcular la mejor potencia de salida para ser usada. Además, el protocolo TPC debe ser ejecutado en hardware que permita las variaciones muy rápidas de potencia de salida. Estas soluciones consiguen reducir las

colisiones y mejorar la capacidad en MANET's. Sin embargo, en WSN, la principal preocupación es la eficiencia en el consumo de energía. Además, los protocolos en WSN están limitados por los recursos disponibles de memoria, la CPU y la radio. Por lo tanto, lo desarrollado para MANET's no siempre (y en este caso no lo es) es aplicable a las redes de sensores.

Capítulo 4

Implementación del TPC en S-MAC

Como ya se explicó anteriormente, los sensores elegidos para la realización del proyecto fueron los MICA2 de la empresa Crossbow [8]. Dichos sensores funcionan con un sistema operativo llamado TinyOS [6] pensado especialmente para su uso en nodos de redes de sensores [5]. La programación de aplicaciones para dicho sistema operativo se realiza en un lenguaje llamado nesC, este lenguaje no es más que una extensión del lenguaje de programación C, diseñada para incluir los conceptos estructurales y de ejecución del modelo presentado por el sistema operativo TinyOS [12].

Para que fuese factible realizar una implementación real del protocolo S-MAC con el TPC integrado la mejor opción era sin duda buscar una implementación ya hecha del protocolo y, sobre ella, añadir el mecanismo de control de potencia; de otra manera la duración y dificultad del proyecto harían inviable su realización puesto que incluiría el desarrollo desde cero de un protocolo nada trivial [15] además de la integración en el mismo del mecanismo de TPC. Por todos estos motivos decidimos buscar una implementación ya realizada y probada de S-MAC en los sensores.

4.1. Implementaciones de S-MAC

En realidad, S-MAC es un protocolo relativamente joven, ya que está diseñado para una tecnología igualmente *joven*. Es por esto que, al menos de manera pública, no abundan las implementaciones de dicho protocolo y la única disponible (al menos al comienzo de este proyecto) es la realizada por Wei Ye. Wei Ye es profesor adjunto de investigación del departamento de Ciencias de la computación y experto en informática en el *Information Sciences Institute* (ISI), de la *University of Southern California* (USC). En su artículo *An Energy-Efficient MAC Protocol for Wireless Sensor Networks* [15] define y explica el funcionamiento del protocolo en cuestión así como una pequeña comparativa de su rendimiento energético respecto a otros protocolos MAC inalámbricos como el implementado por el estándar IEEE 802.11x. Además de este interesante artículo, en la página web del autor [13] en la USC podemos encontrar enlaces para la descarga de la implementación de S-MAC, tanto para TinyOS como para NS-2.*

*NS-2 es un simulador de redes de eventos discretos utilizado principalmente en ambientes académicos debido a su licencia libre (GPLv2) y a la abundancia de documentación en línea.

La implementación de S-MAC realizada por Wei Ye también puede encontrarse (y de hecho el autor lo recomienda como fuente para la descarga [14]) en el repositorio CVS oficial de TinyOS [6]. La versión utilizada en este proyecto es la 1.2, publicada por el autor con fecha del 15 de Mayo de 2004. Dicha versión era la última estable a la hora de comenzar este proyecto aunque en la actualidad existen algunas versiones posteriores accesibles, como ya se dijo, desde el repositorio CVS de TinyOS [14], pero no han sido usadas en este proyecto.

4.2. El S-MAC de la USC

La implementación creada por el profesor de la USC es bastante funcional aunque no dispone de demasiada documentación publicada (aunque la documentación aumenta con el tiempo y, por supuesto, al momento de terminar este proyecto es más abundante que cuando se empezó). Por este motivo los comienzos con el protocolo no fueron nada fáciles ya que nos enfrentamos a la implementación de un protocolo muy complejo escrito por otra persona y en un lenguaje nuevo. Tras unos meses de análisis del protocolo y después de la lectura de diversa documentación, la estructura utilizada por el autor empezó a vislumbrarse. A la hora de gestionar los mensajes, S-MAC utiliza su propio tipo de mensajes, dejando a un lado la mayoría de las interfaces proporcionadas por TinyOS, reescribiendo la mayoría de ellas y utilizando solamente, las de más bajo nivel para el acceso al chip de la radio. El acceso al chip de radio no es algo trivial puesto que supone acceder a un chip concreto para cada tipo de sensor, el hecho de tener que especificar una interfaz concreta para cada chip en lugar de utilizar una interfaz estándar para acceder a los *chip de radio*, hace que esta implementación solamente pueda ser ejecutada en los sensores MICA y MICA2 (los que actualmente utilizan el chip CC1000 de Chipcon [11]. Este chip varía en el caso de los MICAZ lo que los hace incompatibles con esta implementación.

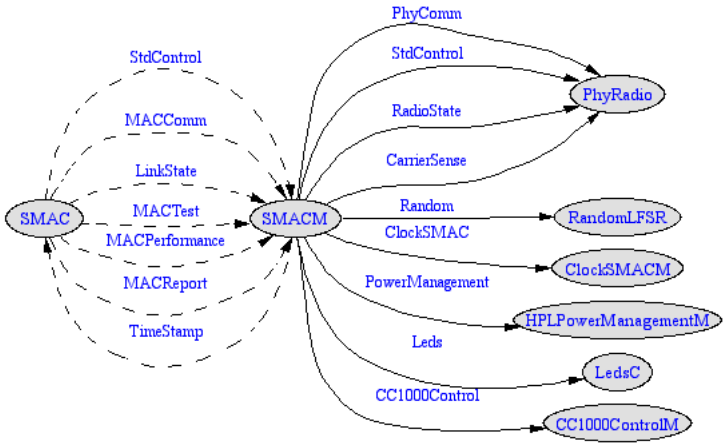


Figura 4.1: Esquema de relaciones entre componentes de S-MAC

Como podemos observar en la figura 4.1, S-MAC utiliza la orientación a componentes aprovechando la versatilidad que ofrece nesC. El protocolo se comunica directamente con algunos componentes de los proporcionados por TinyOS (véase *LedsC*, *Ran-*

domLFSR...) y conecta también con otros componentes implementados por S-MAC (véase *PhyRadio*, *ClockSMACM*...). Si nos fijamos, el módulo *SMACM* utiliza el componente *CC1000ControlM* a través de la interfaz *CC1000Control* pero, el que realmente se relaciona con las capas más bajas de la radio es *PhyRadio*, como muestra la figura 4.2 *PhyRadio* es el encargado de todo lo relacionado con la capa física de la radio y es el que en última instancia (a ojos del componente *SMACM*) se encarga de todo. La conexión con este componente (*CC1000ControlM*) directamente desde *SMACM* se debe a la necesidad de modificar la potencia de transmisión pero eso será explicado más adelante. El funciona-

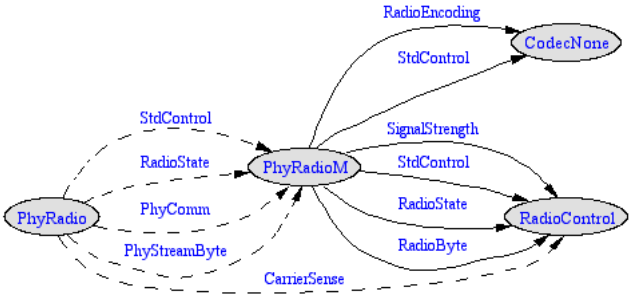


Figura 4.2: Esquema de relaciones del componente PhyRadio

miento de S-MAC descrito en el apartado 3.3 se ajusta con bastante precisión al desarrollo realizado por Wei Ye y sus compañeros. A continuación se procede a explicar los pasos seguidos para la implementación del mecanismo de control de potencia.

4.3. Primeros pasos

Como se comentó en la sección anterior, el componente *SMACM* mostrado en la figura está directamente conectado (*wired*) con el componente *CC1000ControlM* a través de la interfaz *CC1000Control*. Este juego de palabras, lo que viene a significar es que el componente *SMACM* utiliza la interfaz *CC1000Control*, pero una interfaz solamente es una declaración de principios por lo que alguien debe implementarla y ese 'alguien' es el componente *CC1000ControlM*. Observando la figura 4.1 se puede entender mejor el significado de lo expuesto. Esta *conexión* entre componentes se debe a que es el componente *CC1000ControlM* el que ofrece el método *setRFPower(newPower)*. Este método es el encargado de modificar en tiempo de ejecución la potencia de transmisión y para poder realizar llamadas a dicho método, es necesario incluir en la cabecera del componente *SMACM* que vamos a utilizar (*uses*) la interfaz CC1000Control. Una vez hecho esto, desde el componente en cuestión podemos realizar llamadas tal que así:

```
call CC1000Control.SetRFPower(newPower);
```

Modificar la potencia de transmisión en tiempo de ejecución ya es posible desde *SMACM*, ahora es necesario obtener una medida del nivel de *Signal / Noise* (S/N) para, con un nivel umbral prefijado, modificar la potencia para obtener valores en torno a dicho umbral. En otras palabras, se hace necesario conseguir, una vez más en tiempo de ejecución, los valores tanto de potencia de señal recibida como de ruido.

Llegados este punto es conveniente recordar un poco la implementación del TPC realizada en este proyecto. Existen diversas maneras de implementar un mecanismo de control de potencia y en este proyecto se ha elegido una de las muchas posibles. En la implementación elaborada, se aprovecha el flujo normal de mensajes de control entre emisor y receptor [15] para obtener una referencia que permita aproximar la calidad del enlace (distancia entre nodos, ruido existente...) y en función de ello variar la potencia de transmisión. Dicho de otra manera, cuando un nodo quiere transmitir (nodo A) envía al nodo destino (nodo B) un mensaje de control RTS y el nodo B, al recibirlo, si está libre (es decir, si no tiene ninguna otra petición) le contestará con un mensaje *Clear To Send* (CTS) y es justo en el momento, cuando el nodo A recibe ese mensaje (CTS), cuando éste mide la potencia de señal recibida. Los mensajes de control (RTS, CTS, ACK, SYN) se envían todos a la potencia nominal (la máxima en nuestro caso) con lo que se sabe la potencia la que se emitieron y se conoce también la potencia con la que se reciben. De este modo, los únicos paquetes que ven modificada su potencia de transmisión son los paquetes de datos. Este cambio de potencias no supone a la larga un gasto desmesurado puesto que los paquetes más largos suelen ser los de datos por lo que, el mayor consumo se da en su transmisión.

Ahora que se tiene claro el funcionamiento del TPC, se procede a descubrir su implementación sobre el S-MAC de la USC. Para empezar es necesario que desde el propio componente *SMACM* se pueda acceder a los niveles tanto de señal (RSSI) como de ruido. Para ello, es necesario comprobar como se puede tener acceso al canal 0 del ADC ya que es ahí donde el MICA2 obtiene el valor del RSSI. Si exploramos un poco el código de S-MAC, encontramos el componente *PhyRadioM* [tos.system.PhyRadioM]. Este componente podría decirse que, para S-MAC es la capa física que se encarga de enviar y recibir paquetes (codificando y decodificando los bytes). Además puede (opcionalmente) comprobar el CRC de los paquetes recibidos y hacer de interfaz con el control de la radio y la detección física de la portadora. Este componente calcula un promedio del nivel de ruido detectado, es decir, cada vez que termina de recibir un paquete, toma una medida en el canal 0 del ADC y ese valor, lo suma con el anteriormente almacenado y lo divide por 2. Es de alguna manera similar a la media solo que los últimos valores son más influyentes en el resultado. Esta característica que viene ya implementada, sirve perfectamente para el propósito de este proyecto puesto que, ante cambios bruscos en el nivel de ruido, el hecho de que el nivel obtenido sea la media no influye puesto que ese valor se modifica rápidamente ante cambios importantes.

```
async event result_t SignalStrength.sampleReady(uint16_t value)
{
        // will take care of the situation that MAC tries to turn off the
        // radio before sampling of noise level is done
        if (value == 0) return FAIL;
        if (typeRSSI == SIGNAL) { // RSSI of a receiving packet
                ((PhyPktBuf*)recvPtr)->info.strength = value; // put into pkt
                phyUartDebug_byte((uint8_t)(value>>8));
                phyUartDebug_byte((uint8_t)value);
                // don't update average signal strength until CRC is checked
        } else { // measuring noise floor
                // average with previous measurement
                noiseLevel = (noiseLevel + value) >> 1;
                phyUartDebug_byte((uint8_t)(noiseLevel>>8));
                phyUartDebug_byte((uint8_t)noiseLevel);
        }
}
```

Como se puede apreciar en el código que desarrolla el evento *sampleReady(uint16_t value)*, pertenciente a la interfaz *SignalStrength*, la variable noiseLevel actualiza su valor sumándose a si misma con el valor recibido del ADC y desplazando un bit hacia la derecha (o lo que es lo mismo dividiendo por dos). Para poder recuperar ese valor desde *SMACM* se modificó una de las interfaces que conectan dicho componente con *PhyRadioM*, la interfaz *PhyComm*. En dicha interfaz se añadió el método *getNoiseLevel()* que, implementado después en el componente *PhyRadioM* hace accesible a *SMACM* el nivel de ruido calculado en la capa física. De esta manera, desde *SMACM* es posible obtener el nivel medio de ruido del sistema con una simple llamada al método proporcionado por el componente.

```
command uint16_t PhyComm.getNoiseLevel()
{
        return noiseLevel;
}
```

Tras añadir el desarrollo de este simple método en el componente *PhyRadioM*, ya solo faltaba tener acceso al nivel de RSSI, para ello basta con mirar la estructura de los mensajes que se envían y reciben en S-MAC:

```
// Physical-layer header to be put before data payload
typedef struct {
        uint8_t length; // length of entire packet
}__attribute__((packed)) PhyHeader;

// packet information to be recorded by physical layer
typedef struct {
        uint16_t strength;
        uint32_t timestamp;   // can be used w/ external counter of fine resolution
        uint32_t timeCoarse;  // S-MAC system time w/ resolution of 1 ms
}__attribute__((packed)) PhyPktInfo;

// Physical layer packet buffer (for receiving packets)
// Sending buffer should be provided by the top-level application

#define PHY_MIN_PKT_LEN (sizeof(PhyHeader) + 2)
#define PHY_MAX_PAYLOAD (PHY_MAX_PKT_LEN - PHY_MIN_PKT_LEN)

typedef struct {
        PhyHeader hdr;
        char data[PHY_MAX_PAYLOAD];
        int16_t crc;        // last field of a packet
        PhyPktInfo info;  // not part of a packet
} __attribute__((packed)) PhyPktBuf;
```

Los paquetes se encapsulan en la estructura *PhyPktBuf*, como se aprecia en el código anterior, esta estructura está compuesta a su vez de otra llamada *PhyPktInfo* y, dentro de esta podemos ver una variable, *uint16_t* llamada *strength*. Bien, esta variable, cada vez que se recibe un paquete, almacena el valor del RSSI de ese paquete en dicho campo por lo que, una vez tenemos el paquete (desde cualquier componente) podemos acceder a él y conocer el valor del RSSI. De este modo, basta con incluir en el componente *SMACM* una llamada como la que se muestra a continuación para obtener el valor del RSSI para cada paquete recibido:

```
uint16_t rssi;

rssi = ((PhyPktBuf*)packet)->info.strength;
```

Una vez realizado esto, ya son accesibles tanto el nivel de ruido como de señal recibida (RSSI) desde el componente *SMACM*, además también es posible, como ya se comentó anteriormente modificar en tiempo de ejecución desde este mismo componente la potencia de transmisión de los paquetes por lo que ya están listos todos los pasos para proceder a implementar el TPC en S-MAC.

4.4. Implementación real del TPC sobre S-MAC

Una vez comentados todos los 'detalles' arriba relatados, el desarrollo del TPC sobre S-MAC es relativamente fácil. Hay que tener en cuenta que al estar trabajando con los sensores, la programación (y sobre todo su depuración) no es una cuestión trivial por lo que se hace necesario desde el primer momento desarrollar/encontrar una manera de poder depurar. De ese modo y vistas las limitaciones presentadas por los dispositivos, la opción más factible parece hacerlo a través del puerto serie.

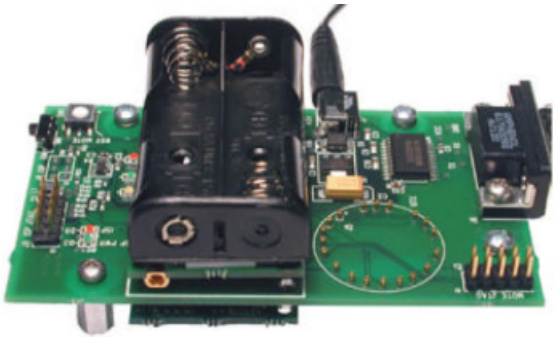


Figura 4.3: La plataforma de programación también permite que el MICA2 se comunique con el PC

Existen varias maneras de transmitir bytes por el puerto serie desde los MICA2 pero todas ellas a muy bajo nivel. La solución más sencilla resultó venir implementada en el propio S-MAC. Se trata de una macro que los autores de S-MAC desarrollaron para depurar su código. Esta macro resultó ser bastante útil aunque un poco incómoda puesto que solo permite enviar un byte por cada llamada. Aún así y tras varias pruebas fallidas se decidió utilizar el método proporcionado por el propio S-MAC. Si se mira en *contrib/s-mac/tos/system* encontramos un fichero llamado *smacUartDebug.h*, en su interior está la definición de varias macros entre las que se encuentra *SMAC_UART_DEBUG_BYTE*.

```
#elif defined SMAC_UART_DEBUG_BYTE
// Debug by sending a byte through UART

#include "uartDebug.h"

static inline void smacUartDebug_init()
{
   uartDebug_init();
}

static inline void smacUartDebug_state(char byte)
{
}
```

```
15 static inline void smacUartDebug_event(char byte)
   {
   }

   static inline void smacUartDebug_byte(char byte)
20 {
       uartDebug_txByte(byte);
   }
```

Como se puede observar, definiendo la macro mencionada el método para transmitir bytes a través del puerto serie *smacUartDebug_byte(char byte)* queda implementado y listo para su uso. A partir de entonces, cualquier lugar del código donde se sitúe hará que se envíe, a través de la interfaz UART, el byte correspondiente. Lo primero que se desarrolló, a fin de comprobar el correcto funcionamiento del protocolo fue un sencillo depurador que, cada vez que se enviaba y recibía un paquete enviaba al puerto serie el paquete (su nombre); si el paquete es enviado aparece su nombre y si por el contrario se trata de un paquete recibido aparece también su nombre, pero seguido de un '*'. Para poder escuchar el puerto serie se empleó la utilidad *hyperterminal*. Conforme fue avanzando

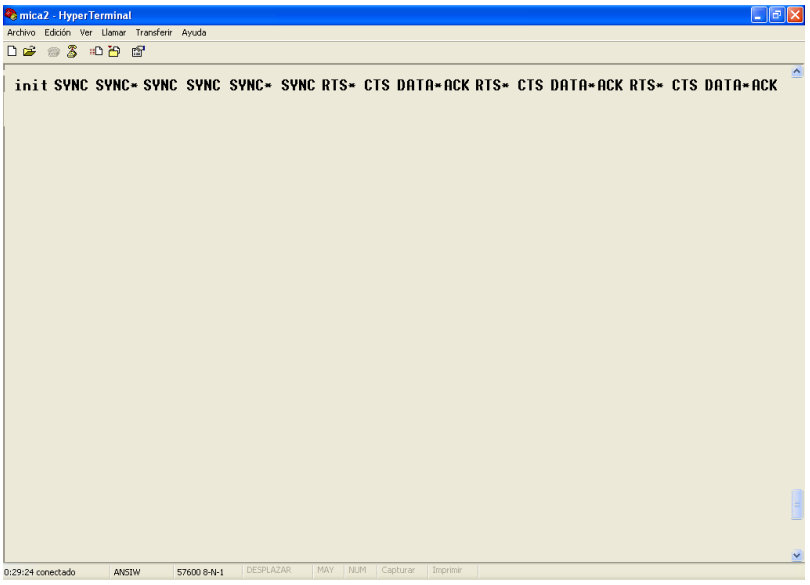


Figura 4.4: Al escuchar el puerto serie se puede ver la secuencia de paquetes

el desarrollo, y una vez comprobado el funcionamiento del protocolo, nuevas formas de depuración se hicieron necesarias. Se desarrollaron dos tipos más y se separaron mediante el uso de macros, de tal manera que desde un fichero de configuración es posible definir si se quiere activar la depuración y, en tal caso, que tipo de depuración se quiere. Es posible definir si se quiere una depuración por paquetes (como la que muestra la figura 4.4) o una depuración más relacionada ya con el control de potencia, desde el punto de vista del transmisor o del receptor (ver en la sección 4.5 el fichero *config.h*).

Una vez conseguido un buen depurador y recordando que ya es posible cambiar la potencia de transmisión así como acceder a los niveles potencia de señal recibida y de ruido desde el componente SMACM, solo falta aglutinar todos los elementos para hacer que S-MAC funcione con el TPC. Si recordamos la sección 2.1.3 y las ecuaciones 2.1 y 2.2, el paso de los valores leídos en el ADC a valores en dBm se realiza mediante unas sencillas

operaciones. A continuación se muestra el código donde se realiza la implementación del TPC:

```
    void handleCTS(void* pkt)
    {
       // internal handler for CTS
#ifdef SMAC_TPC
#ifdef SMAC_UART_DEBUG_TX
        char *dato;
#endif
        uint16_t senalInt, ruidoInt;
#endif
        MACCtrlPkt* packet;
        packet = (MACCtrlPkt*)pkt;
        call Leds.greenToggle();

#ifdef SMAC_TPC
        rssi = ((PhyPktBuf*)packet)->info.strength;
        //el valor del RSSI medido en el canal 0 del conversor A/D
        noiseLevel = call PhyComm.getNoiseLevel();
        //el valor medio del ruido proporcionado por la capa física

        /*********************************************************************
        Para pasar a dBm el valor leído del conversor A/D hay que realizar
              una fórmula (varía dependiendo de si es 433MHz o 915MHz):
                                    ____________________
                                    Vrssi = Vbatt*ADCCounts/1024
                                    ____________________________________
                          -->       RSSIdbm = -51.3*Vrssi - 49.2 [dBm] @ 433/315 MHz
              MICA2       -->
                          -->       RSSIdbm = -50.0*Vrssi - 45.5 [dBm] @ 915 MHz
        *********************************************************************/

        senal = -51.3*3*((float)rssi/1024)-49.2;
        ruido = -51.3*3*((float)noiseLevel/1024)-49.2;

       //Pasamos a dBm los valores devueltos por el canal 0 del ADC
       //              tanto de señal como de ruido.

        snr = senal-ruido;

        senalInt = senal;
        ruidoInt = ruido;
        snrint = snr;

#ifdef SMAC_UART_DEBUG_TX
        /*********************************************************************
                                        SEÑAL / RUIDO (dBm)
        *********************************************************************/
        dato = intoa(snrint,10);
        smacUartDebug_byte('S');
        smacUartDebug_byte('/');
        smacUartDebug_byte('N');
        smacUartDebug_byte('(');
        smacUartDebug_byte('R');
        smacUartDebug_byte(')');
        smacUartDebug_byte('=');

        if(snrint<0)
              smacUartDebug_byte('-');
        smacUartDebug_byte(dato[0]);
        if(snrint>9)
              smacUartDebug_byte(dato[1]);
        if(snrint>99)
              smacUartDebug_byte(dato[2]);
        smacUartDebug_byte('␣');

        /*************************************************************
```

```
                                        SEÑAL (dBm)
            ************************************************************/

            dato = intoa(senalInt,10);
            smacUartDebug_byte('S');
            smacUartDebug_byte('E');
            smacUartDebug_byte('N');
            smacUartDebug_byte('=');


            if(senal<0)
                  smacUartDebug_byte('-');
            smacUartDebug_byte(dato[0]);
            if(senalInt>9)
                  smacUartDebug_byte(dato[1]);
            if(senalInt>99)
                  smacUartDebug_byte(dato[2]);
            smacUartDebug_byte('␣');

            /*********************************************************
                                    RUIDO (dBm)
            *********************************************************/

            dato = intoa(ruidoInt,10);
            smacUartDebug_byte('R');
            smacUartDebug_byte('U');
            smacUartDebug_byte('I');
            smacUartDebug_byte('=');

            if(ruido<0)
                  smacUartDebug_byte('-');
            smacUartDebug_byte(dato[0]);
            if(ruidoInt>9)
                  smacUartDebug_byte(dato[1]);
            if(ruidoInt>99)
                  smacUartDebug_byte(dato[2]);
            smacUartDebug_byte('␣');
#endif
#endif
            if (packet->toAddr == TOS_LOCAL_ADDRESS) {
            if (state == WAIT_CTS && packet->fromAddr == sendAddr) {
                  // cancel CTS timer
                  geneTime = 0;
                  // schedule sending DATA
                  state = TX_PKT;
                  howToSend = SEND_DATA;
                  txDelay = SIFS;
                  smacUartDebug_state(state);

#ifdef SMAC_UART_DEBUG_PKT
                  smacUartDebug_byte('C');
                  smacUartDebug_byte('T');
                  smacUartDebug_byte('S');
                  smacUartDebug_byte('*');
                  smacUartDebug_byte('␣');
#endif
             } else {
                  handleErrPkt();
             }
        } else { // packet destined to another node
            if (state == DATA_SENSE1 || state == DATA_SENSE2) {
                  geneTime = 0;
            }
            if (state == IDLE || state == BACKOFF ||
              state == DATA_SENSE1 || state == DATA_SENSE2) {
                  UPDATE_NAV(packet->duration);
                  sleep(FORCE);  // avoid overhearing other's packet
                  if (nav == 0) nav = 1;  // in case duration is 0 by mistake
```

```
            }
        }
    }
```

Se puede observar, al principio del código, como se condiciona gran parte de la implementación a la definición de la macro *SMAC_TPC*, de este modo, es posible desde un archivo de configuración y comentando o descomentando una línea es posible activar o desactivar el TPC de S-MAC. La definición de esta macro así como las de los tres tipos de depuradores hacen que, desde un fichero de configuración que puede estar integrado en el de la propia aplicación que utilice S-MAC.

En el método arriba mostrado (*handleCTS()*) se obtiene la medida de S/N pero todavía no se ha modificado la potencia de transmisión; ésta se debe modificar únicamente cuando un mensaje de datos se vaya a enviar y, acto seguido debe volver de nuevo a su valor por defecto (la potencia máxima en nuestro caso), por este motivo se realiza el cambio de potencia en el método (*sendDATA()*). A continuación se muestra el código de dicho método, con la implementación realizada:

```
void sendDATA()
   {
      // send a unicast data packet
      // assume all MAC header fields have been filled except the duration

#ifdef SMAC_UART_DEBUG_TX
          char *dato;
          uint8_t potencia = 0;
#endif
#ifdef SMAC_TPC
          uint8_t newPower = 0xFF;       //antes de realizar los cálculos se le
          uint8_t defaultPower = 0xFF;   //da el valor por defecto, por si no se
          snDif = 0;                     //puede tomar la medida la primera vez.


/*****************************************************************************
Aquí cambiamos la potencia de transmisión según lo que valga snDif que no es ni
más ni menos que la diferencia entre la S/N umbral que hemos definido y la S/N real
es decir:              snDif = [(S/N)real] - [(S/N)umbral]
Según lo que valga snDif, tendremos que, a la potencia máxima de transmisión restarle
lo que valga snDif, así, si la potencia máxima es 10 dBm y snDif vale 15 habrá que
establecer la potencia de transmisión a -5dBm. Los 31 posibles valores para la
potencia de tx están almacenados en el array poTx[ ] en orden descendente, es decir,
la posición 0 contiene la máxima potencia y la posición 30 la mínima. Como cada
posición que avanzamos resta 1dB solo habrá que acceder al array en la posición que
indique snDif y tendremos el valor del registro para la nueva potencia de tx.
*****************************************************************************/

          if(snrint>SMAC_SNR_UMBRAL)
                snDif = snrint - SMAC_SNR_UMBRAL;
          if(snDif>30)           //A la máxima potencia (0xFF) puedes restarle como
                snDif = 30;      //mucho 30dB, es el mínimo que ofrece CC1000

          newPower = poTx[snDif];
          numMsgTx++;    //indica el número de mensajes transmitidos

#ifdef SMAC_UART_DEBUG_TX
          dato = intoa(snDif,10);
          smacUartDebug_byte('S');
          smacUartDebug_byte('/');
          smacUartDebug_byte('N');
          smacUartDebug_byte('(');
          smacUartDebug_byte('d');
          smacUartDebug_byte(')');
          smacUartDebug_byte('=');
          smacUartDebug_byte(dato[0]);
```

```
            if(snDif>9)
                smacUartDebug_byte(dato[1]);
            if(snDif>99)
                smacUartDebug_byte(dato[2]);
            smacUartDebug_byte('␣');
#endif
            call CC1000Control.SetRFPower(newPower);  //la nueva potencia de tx
#endif

#ifdef SMAC_UART_DEBUG_TX
#ifdef SMAC_TPC
            dato = intoa(SMAC_SNR_UMBRAL,10);
            smacUartDebug_byte('S');
            smacUartDebug_byte('/');
            smacUartDebug_byte('N');
            smacUartDebug_byte('(');
            smacUartDebug_byte('U');
            smacUartDebug_byte(')');
            smacUartDebug_byte('=');

            smacUartDebug_byte(dato[0]);
            if(SMAC_SNR_UMBRAL>9)
                smacUartDebug_byte(dato[1]);
            if(SMAC_SNR_UMBRAL>99)
                smacUartDebug_byte(dato[2]);
            smacUartDebug_byte('␣');
#endif
            potencia = call CC1000Control.GetRFPower();
            dato = intoa(potencia,16);
            smacUartDebug_byte('P');
            smacUartDebug_byte('T');
            smacUartDebug_byte('X');
            smacUartDebug_byte('=');
            smacUartDebug_byte('0');
            smacUartDebug_byte('x');
            smacUartDebug_byte(dato[0]);
            if(potencia>0xF)
                smacUartDebug_byte(dato[1]);
            smacUartDebug_byte('\n');
            smacUartDebug_byte(0xD);       //retorno de carro
#endif

#ifdef SMAC_TX_TIME_STAMP
        dataPkt->txTimeStamp = clockTime;
#endif
        // neighbNav tracks the time I have left for tx
        dataPkt->duration = neighbNav - durDataPkt;
            call Leds.redToggle();
        call PhyComm.txPkt(dataPkt, txPktLen);
#ifdef SMAC_TPC
        call CC1000Control.SetRFPower(defaultPower);
//Una vez que se envia el pkt se establece otra vez la potencia de tx por defecto
#endif
            radioState = RADIO_TX;
        txErrTime = TX_PKT_DONE_TIME;

#ifdef SMAC_UART_DEBUG_PKT
            smacUartDebug_byte('D');
            smacUartDebug_byte('A');
            smacUartDebug_byte('T');
            smacUartDebug_byte('A');
            smacUartDebug_byte('␣');
#endif
    }
```

Las macros de depuración arriba mentadas se pueden encontrar también en el código mostrado para *sendDATA()*. Además, se pueden encontrar las dos llamadas al método

SetRFPower() de *CC1000Control*, una con la nueva potencia de transmisión calculada (*newPower*) y la otra con la potencia de transmisión por defecto (la máxima en nuestro caso, llamada en el código *defaultPower*. En cuanto a la potencia de transmisión por defecto se encontraron algunos problemas que se comentan más detalladamente en la sección 7.1.

4.5. SMACTest

La implementación del protocolo no viene sola sino que trae consigo unas cuantas aplicaciones de ejemplo que son ejecutables desde el primer momento, una de ellas es *SMACTest*. Esta aplicación testea la funcionalidad básica del protocolo S-MAC probando el envío y recepción de mensajes. La configuración, por defecto, es para ser usada con 3 nodos sensores. Cada nodo envía 10 mensajes broadcast y 10 mensajes unicast, cada uno con 5 fragmentos (los unicast). Esta configuración mencionada es modificable desde el archivo *config.h*. También se puede modificar S-MAC para ser ejecutado en diferentes modos: *"low duty cycle with adaptative listen"*, *"low duty cycle without adaptive listen"* y *"fully active mode"*. Además, se incluyeron algunos cambios en el fichero de configuración para hacer accesibles las modificaciones realizadas en la implementación del TPC. Como se puede ver más abajo, este fichero nos permite desde elegir el tipo de mensajes que se enviarán (*broadcast* o *unicast*) hasta cambiar el tamaño de los mensajes.

A continuación se muestra el fichero citado, actualizado a la versión de S-MAC que incluye el TPC y en el que se pueden ver las opciones configurables:

```
#ifndef CONFIG
#define CONFIG

//Configure Physical layer. Definitions here override default values
//Default values are defined in PhyMsg.h
//------------------------------------------------------------------
//#define PHY_MAX_PKT_LEN 100        // max: 250 (bytes), default: 100

//carrier sense threshold to determine a busy channel
//smaller value -> higher threshold -> more aggressive Tx
//this is only for mica2 and mica2dot
//#define RADIO_BUSY_THRESHOLD 0xb5   // 0x60 - 0xff, default: 0xb5

//Configure S-MAC into different operating modes
//-----------------------------------------------
//#define SMAC_DUTY_CYCLE 50         // 1 - 99 (%), default: 10
//#define SMAC_NO_ADAPTIVE_LISTEN   // default: adaptive listen is enabled
//#define SMAC_NO_SLEEP_CYCLE       // default: low duty cycle mode

//With the following macro defined, the node is configured as a slave on
//sleep schedules. It keeps listening to SYNC packets until it receives
//one and adopts it. With one master node and all other nodes as slaves,
//its easy to set up a network with only the master's schedule.
//#define SMAC_SLAVE_SCHED

//User adjustable S-MAC parameters
//--------------------------------
//Definitions here override default values in SMACConst.h

#define SMAC_MAX_NUM_NEIGHB 200   // default value 20
//#define SMAC_MAX_NUM_SCHED 4     // default value 4
//#define SMAC_RTS_RETRY_LIMIT 7   // default value 7
//#define SMAC_DATA_RETX_LIMIT 3   // default value 3
```

```
//the following macro defines the maximum time that S-MAC can hold a message
//for transmission. If it cannot send out the message within the time, it
//will drop the message and signal upper layer tx failure.
//default value: 2min in low duty cycle mode and 10s in fully active mode
//#define SMAC_MAX_TX_MSG_TIME 120000

//The following two macros define the period to search for potential
//neighbors on different schedules. The period is expressed as the
//number of SYNC_PERIODs (10s). Therefore, 30 means after every 30
//SYNC_PERIODs, which is 300s, the node will keep listening for an entire
//SYNC_PERIOD. Maximum value is 255.
//SHORT_PERIOD is used when I have no neighbor -- search more aggressively
//LONG_PERIOD is used when I have neighbors -- don't perform too often
//#define SMAC_SRCH_NBR_SHORT_PERIOD 3  // max: 255, default: 3  (30s)
//#define SMAC_SRCH_NBR_LONG_PERIOD 30  // max: 255, default: 30 (300s)


//Now by default, S-MAC uses timer/counter 0, which conflicts with Clock
//and Timer components. If you want to use Timer or Clock, uncomment the
//following line to let S-MAC use the 16-bit timer/counter 1
#define SMAC_USE_COUNTER_1

//define the following macro to put a time stamp on each outgoing packet
//#define SMAC_TX_TIME_STAMP
//TST_MSG_INTERVAL controls how fast a node generates a message. Setting
//it to 0 makes it generates second packet right after the first is sent.

//Esta macro se utiliza para definir el valor de S/N umbral que se desea,
//es decir, si se está usando S-MAC con TPC, se debe establecer un umbral
//de S/N a partir del cual, se reciban los paquetes correctamente y exista
//una BER aceptable.

#define SMAC_SNR_UMBRAL 16        //por defecto tiene el valor 16

#define SMAC_TPC                  //por defecto está definida

//Configure the test application
//-------------------------------
#define TST_MIN_NODE_ID 1          // at least 2 nodes. node IDs must be
#define TST_MAX_NODE_ID 3          // consecutive from min to max
#define TST_MSG_INTERVAL 200       // in ms

//By default, each node keeps sending until it is powered off.
//To let a node automatically stop after sending sepecified number of
//messages, define the following macro
//#define TST_NUM_MSGS 20

//number of fragments in each unicast test message, max: 8
#define TST_NUM_FRAGS 1

//By default, each node alternate in sending broadcast and unicast
//for unicast, node i sends to (i+1), and node MaxId sends to MinId
//#define TST_BROADCAST_ONLY       // test broadcast only if defined
#define TST_UNICAST_ONLY          // test unicast only if defined
//#define TST_UNICAST_ADDR 1       // specify unicast addr. default one
//#define TST_RECEIVE_ONLY         // set a node that only receives packets

//S-MAC debugging with a snooper
//-------------------------------
//Debug by adding bytes to data pkts, so that snooper can show them
//#define SMAC_SNOOPER_DEBUG

//S-MAC debugging with a serial port (UART)
//-----------------------------------------
//Don't enable it unless you know what's going to happen.
//There is a known problem on Mica2 motes. If UART debugging is enabled
//but the mote is not connected with the serial board/cable, very often
//it fails to start. It occasionally happens when the mote connects with
```

```
//the serial board/cable.

//The following macros are mutally exclusive. You can only define one
//Debugging with predefined S-MAC states and events
//#define SMAC_UART_DEBUG_STATE_EVENT
//Debugging by sending a byte to UART
//#define SMAC_UART_DEBUG_BYTE

//Las siguientes macros se utilizan para depurar, por el puerto serie, las variables
//de transmisión, de recepción o de paquetes respectivamente. Para que funcione
//cualquiera de ellas es necesario que esté definida la macro SMAC_UART_DEBUG_BYTE

//#define SMAC_UART_DEBUG_TX
//Sirve para mostrar los valores de señal, ruido y S/N en el sensor "transmisor"
//es decir, al recibir un CTS y al transmitir los datos.

//#define SMAC_UART_DEBUG_RX
//Esta macro hace lo mismo pero, en el sensor transmisor lo que significa que muestra
//los valores de señal, ruido y S/N al recibir un RTS y al recibir los datos. También se
//muestra la S/N umbral establecida para la implementación del TPC. (Todos los valores
//están expresados en unidades logarítmicas.

//#define SMAC_UART_DEBUG_PKT
//Esta última macro se define cuando se quieren enviar al puerto serie los paquetes que
//se envían o se reciben. Estos últimos van marcados con un '*'.
#endif  // CONFIG
```

Como se puede observar, se incluyen al final una serie de macros que nos permiten depurar diversos aspectos del protocolo a través del puerto serie. Estas macros ya han sido explicadas en la sección 4.4.

La aplicación *SMACTest* original, durante su ejecución, activa o desactiva el LED amarillo para mostrar actividad (ya sea por envío de paquetes, recepción de paquetes, transmisión y recepción de paquetes SYN... El LED rojo cambia de estado cuando un paquete o fragmento en enviado. El LED verde cambia de estado cuando un paquete o fragmento es recibido.

Al incluir el TPC en el S-MAC se incluyó el uso de los LEDs desde el propio componente principal de S-MAC por lo que, al utilizar esta aplicación de test se desactivó el uso de los leds y se redefinió su significado para permitir nuevas formas de depuración. En la aplicación desarrollada en este proyecto se enciende el LED amarillo cada vez que el sensor está despierto (*wakeUp*) y los LEDs rojo y verde se encienden cuando se recibe o se envía un paquete (respectivamente).

Capítulo 5

Implementación del circuito de medidas de consumo

Para la realización del circuito de medidas de consumo se plantearon diversas opciones (ver sección 7.2) pero al final se optó por realizar un circuito impreso que realizara las medidas. Este circuito impreso consta de un circuito integrador realizado con un amplificador operacional y diversos componentes así como de las entradas/salidas necesarias para alimentarlo, alimentar el MICA2 y además proporcionar una salida desde la que se toma la medida de consumo.

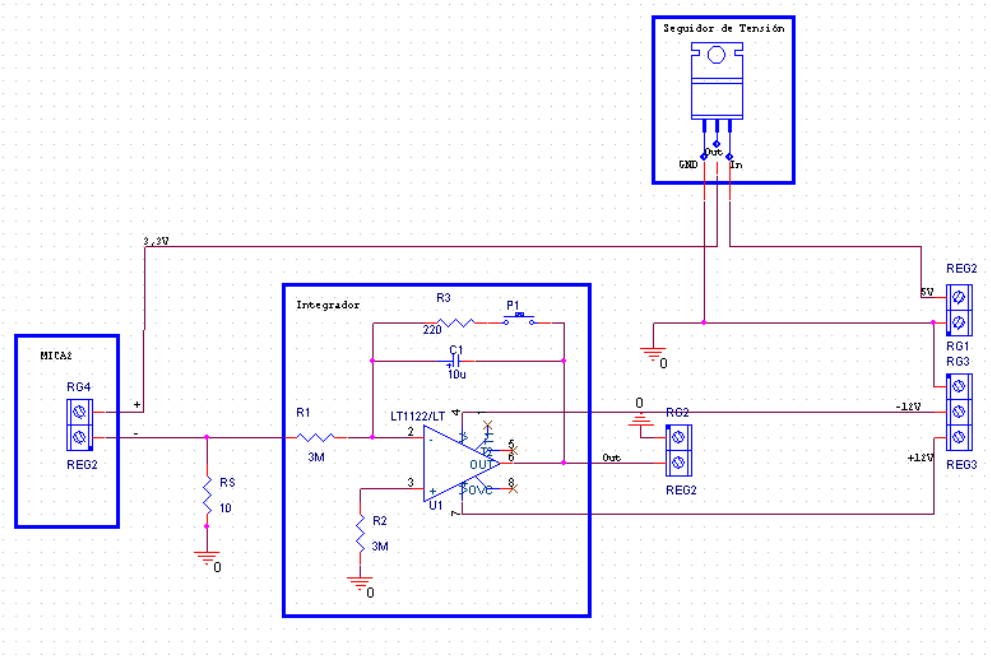


Figura 5.1: Esquemático del PCB construido para tomar las medidas

Como se puede ver en la figura 5.1 el circuito dispone también de un seguidor de tensión que ofrece a su salida una tensión constante de 3,3V, de este modo podemos alimentar desde el propio circuito de medida a los MICA2 y asegurándo que reciben

una tensión constante. Así, como se puede observar en la figura 5.1 el circuito necesita ser alimentado por tres tensiones: ±12V y 5V. Los ±12V se utilizan para alimentar el operacional presente en el circuito y los 5V son la entrada al seguidor de tensión que, a su salida proporciona los 3,3V necesarios para los sensores.

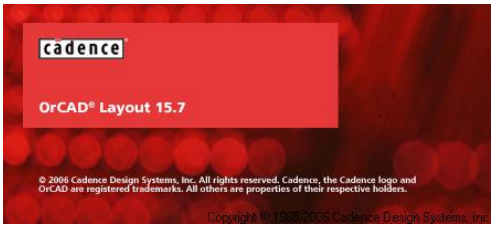


Figura 5.2: El programa utilizado para realizar el PCB fue el Orcad Layout

A la hora de pasar el esquemático a circuito impreso (PCB) se utilizó la herramienta *Orcad Layout*, de la empresa *Cadence* desde un PC de la universidad que disponía de la licencia correspondiente. Este programa dispone de diversas herramientas de *enrutado* automático aunque también permite su realización de manera manual. Tras diversas pruebas se consiguió pasar desde el esquemático realizado en *Capture CIS* (otro programa de la misma empresa distribuido a la vez en una sola *suite*) mostrado en la figura 5.1 hasta el PCB sin enrutar.

Una vez conseguido, solo es cuestión de modificar la posición de los elementos para optimizar las conexiones y que, de ese modo las pistas sean lo más cortas posibles así como que no se crucen o no formen ángulos de 90°. De este modo y tras un largo período de tiempo se consiguió un resultado que se muestra en la figura 5.3

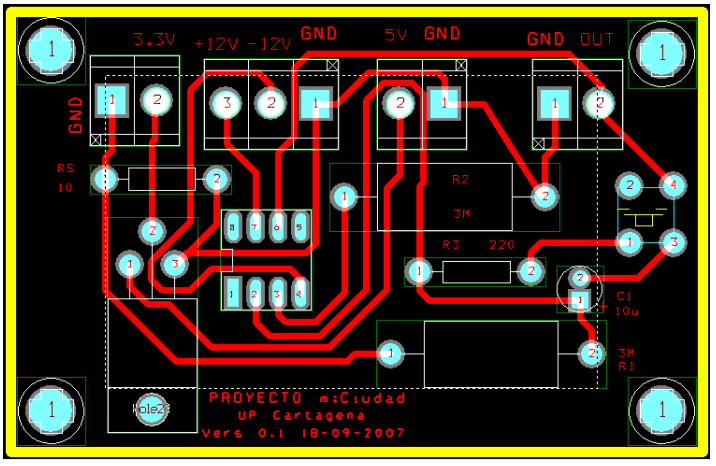


Figura 5.3: Una captura del layout del circuito

Este circuito se presenta como la mejor opción para la realización de las pruebas de medida de consumo. La mejor opción puesto que se trata de un diseño probado (la implementación del mismo se probó primero sobre una placa board, ver figura 7.6), un diseño compacto (la placa es sólo un poco más grande que los sensores) y más seguro (la alimentación a los dispositivos se realiza a través de un seguidor de tensión, lo que garantiza

una salida constante de 3,3V sea cual sea la tensión proporcionada por la fuente de alimentación). Además, el hecho de disponer del circuito en un PCB reduce la probabilidad de fallo puesto que, al estar los componentes soldados y las conexiones hechas mediante pistas es más dificil que se produzca un fallo. En las primeras pruebas realizadas con el mismo circuito (el circuito integrador sin seguidor de tensión) sobre la placa board eran constantes las desconexiones de componentes en los traslados, así como los malos contactos que producían algunos componentes al recibir tirones o desplazamientos fruto de el amplio cableado presente.

Capítulo 6

Pruebas y resultados

El objetivo de las mediciones era el de cuantificar el consumo energético que los sensores tenían tras realizar la modificación del protocolo (implementación del TPC en S-MAC). El objetivo era comparar estos resultados con los obtenidos por un sensor en las mismas condiciones pero con el protocolo S-MAC original. En principio esta medida se puede obtener de manera sencilla colocando un medidor de corriente (amperímetro) en serie con el aparato, tomando muestras durante un tiempo determinado y realizando luego los cálculos pertinentes:

$$I = \frac{V}{R}; \tag{6.1}$$

$$P = V \cdot I; \tag{6.2}$$

$$E = P \cdot t; \tag{6.3}$$

Por lo que, una vez tomadas las muestras solo queda multiplicarlas todas por la tensión de alimentación del aparato (como se indica en 6.2 y por el tiempo que haya durado la prueba para obtener una medida de la energía consumida (ver 6.3. De este modo, aplicando esta prueba a dos sensores durante el mismo tiempo, uno con una aplicación de test que utilice S-MAC con TPC y otro sin TPC podemos obtener una relación que nos indique el ahorro energético producido.

Para que la tensión suministrada fuese constante la mejor opción parecía alimentar el MICA2 con una fuente de alimentación para, de este modo, obtener realmente una tensión constante. En el caso de que se hubiese dejado la alimentación original mediante baterías AA hubiera surgido un problema ya que, a lo largo del tiempo, con el desgaste de la pila, la tensión suministrada decrece y, al no ser constante, el cálculo de la energía consumida se hace más complejo. Por este motivo decidimos utilizar una fuente de alimentación para proporcionar energía a los sensores y que estos estuviesen alimentados de forma constante.

6.1. Medidas de consumo

Las primeras medidas de consumo energético en los MICA2 se realizaron en el interior del laboratorio I+D IT2. Para comenzar las pruebas se decidió hacer unas pruebas iniciales que permitieran comprobar la aplicación de test (ver 4.5) y comenzar a comparar consumos respecto a la misma aplicación haciendo uso de S-MAC sin TPC. Para ello se montó una prueba en el laboratorio, separando los sensores 50 cm. La aplicación de test está programada para que el nodo 1 envie mensajes unicast cada 500 ms al nodo 2; éste al recibirlos contesta con un ACK y se queda a la espera de más mensajes. De las primeras pruebas con el circuito integrador comprobamos que, con la tensión de alimentación que tenía el operacional ($\pm$12V) entraba en saturación, aproximadamente al llegar la tensión de salida a los 9V. Sabiendo esto se conectó el sensor al circuito de medida y se dejó funcionar durante 5 min para ver, aproximadamente, qué pendiente tenía la función de salida del integrador, y con ello, comprobar cuán larga podía ser la prueba sin que el operacional entrara en saturación.

Figura 6.1: La primera prueba fue realizada a 50 cm de distancia entre nodos

Tras una primera estimación de la pendiente y para no arriesgar ante posibles cambios en un período más largo, se decidió hacer las pruebas de 25 min de duración. Con esta duración se realizó, con la misma distancia, la medida de consumo para un sensor con TPC primero y después para otro sin él.

Tras estas primeras medidas en el laboratorio, se hicieron otras en un espacio abierto (en el patio del Cuartel de Antigones) y se obtuvieron ahorros cercanos al 12 %, lo cual era un buen ahorro. Tras hacer las pruebas a distancias mayores de 16 metros, se descubrió que los nodos se quedaban sin cobertura, y una de las posibles causas era probablemente el hecho de que estaban a ras del suelo. De este modo se descubrió que las medidas no podían ser consideradas como válidas, pues el hecho de colocar la antena junto al suelo provocaba una variación en su diagrama de radiación, que no permitía establecer unas medidas objetivas, por lo que se consideraron todos los resultados obtenidos como inválidos.

Tras el error cometido en las anteriores se decidió levantar los sensores una distancia sobre el suelo, para conseguirlo se colocaron los nodos encima de sillas. La sujección se

realizó a las mismas con cinta adhesiva de manera que las antenas quedaban totalmente libres. La colocación de la antena se realizó llevando cuidado de que el respaldo de las sillas interfiriera lo mínimo posible con el diagrama de radiación [8].

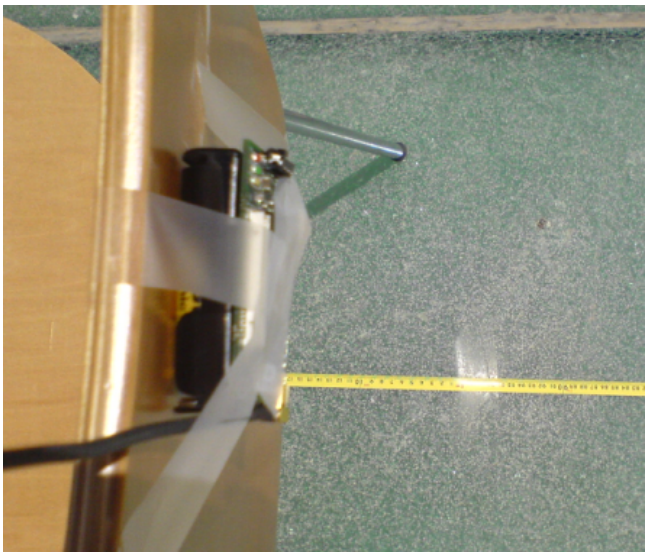


Figura 6.2: Los sensores se fijaron a las sillas mediante el uso de cinta adhesiva

Durante la realización de estar pruebas, se pudo apreciar que existía bastante ruido a la salida del integrador; entonces se decidió tomar como muestra el último milisegundo y, empleando las funciones del osciloscopio, tomar la medida en ese último milisegundo y los valores máximos y mínimos (para tener una idea de cómo el ruido variaba realmente la señal de salida). Tras varias pruebas se determinó que el error introducido por el ruido era, en media, de $\pm$160mV que, pasado a energía es aproximadamente $\pm$1 J por lo que, a la hora de la verdad es un error despreciable. Aún así se decidió seguir esta metodología y tomar como valor válido la media de este último milisegundo, además con cada medida se cuantifica el error.

Figura 6.3: El osciloscopio así como el circuito de medida se sacaron al exterior

Para continuar, se realizaron diversas medidas para distintas distancias pero, a medida que se realizaban más medidas se iba descubriendo que los resultados no siempre eran los esperados.

Figura 6.4: Las pruebas se realizaron para distintas distancias entre nodos

Como es normal, las pruebas se realizaron en distintos días y, haciendo uso de la herramienta de depuración diseñada saltó a la luz que los niveles de ruido, dentro de la universidad variaban mucho (del orden de ± 10 dBm) de un día para otro lo que, en la práctica, se traducía en un consumo muy dispar en las medidas con TPC dependiendo del día. El TPC se guía por el nivel de S/N existente, tomando un nivel como umbral, y reduciendo la potencia de transmisión hasta llegar a ese valor umbral [1]. Si la S/N es muy baja, la potencia de transmisión (y en consecuencia el consumo) aumenta.

A pesar de todo se prosiguió con las medidas y se realizaron pruebas de consumo desde 50cm doblando las distancias (hasta 32m) y una última prueba a 40m. Los resultados se muestran a continuación pero, como se puede observar hay algunas irregularidades.

	Tiempo (min)	Distancia (m)	Vout (Avg) (V)	Maximo	Minimo	Energía consumida (J)	Error (V)	Error (J)
Con TPC	25,01000000	0,50000000	-4,92700000	-4,80000000	-5,04000000	40,16248675	0,24	1,95636225
	25,01000000	0,50000000	-5,08600000	-5,04000000	-5,12000000	41,45857675	0,08	0,65212075
	25,01000000	0,50000000	-4,73400000	-4,72000000	-4,80000000	38,58924544	0,08	0,65212075
	25,01000000	0,50000000	-4,44800000	-4,40000000	-4,48000000	36,25791376	0,08	0,65212075
	25,01000000	1,00000000	-4,16600000	-4,12000000	-4,24000000	33,95918811	0,12	0,97818113
	25,01000000	1,00000000	-5,08800000	-5,08000000	-5,16000000	41,47487976	0,08	0,65212075
	25,01000000	2,00000000	-5,01700000	-4,88000000	-5,12000000	40,89612260	0,24	1,95636225
	25,01000000	4,00000000	-5,11300000	-5,00000000	-5,28000000	41,67866750	0,28	2,28242263
	25,01000000	8,00000000	-5,30500000	-5,12000000	-5,40000000	43,24375730	0,28	2,28242263
	25,01000000	16,00000000	-5,20700000	-5,08000000	-5,36000000	42,44490938	0,28	2,28242263
	25,01000000	32,00000000	-5,19600000	-5,16000000	-5,24000000	42,35524278	0,08	0,65212075
	25,01000000	40,00000000	-4,96700000	-4,92000000	-5,00000000	40,48854713	0,08	0,65212075
Sin TPC	25,01000000	0,50000000	-5,08700000	-5,08000000	-5,12000000	41,46672826	0,04	0,32606038
	25,01000000	1,00000000	-5,11100000	-5,08000000	-5,16000000	41,66236448	0,08	0,65212075
	25,01000000	2,00000000	-5,16700000	-5,12000000	-5,20000000	42,11884901	0,08	0,65212075
					MEDIA DE CONSUMO	41,74931391		
BLINK	25,01000000	0,00000000	-7,94500000	-7,76000000	-8,08000000	64,76374209	0,32	2,60848300

Figura 6.5: Resultados de consumo para las distintas pruebas realizadas

Estas irregularidades pueden ser debidas a diversos factores entre los que se pueden encontrar posibles interferencias en el espectro radio, el factor de corrección de potencia de la fuente de alimentación o incluso alguna mala conexión en algun punto del circuito. Por estos motivos y teniendo en cuenta los resultados, se decidió que las pruebas, a falta de una investigación más profunda, no resultaban del todo concluyentes.

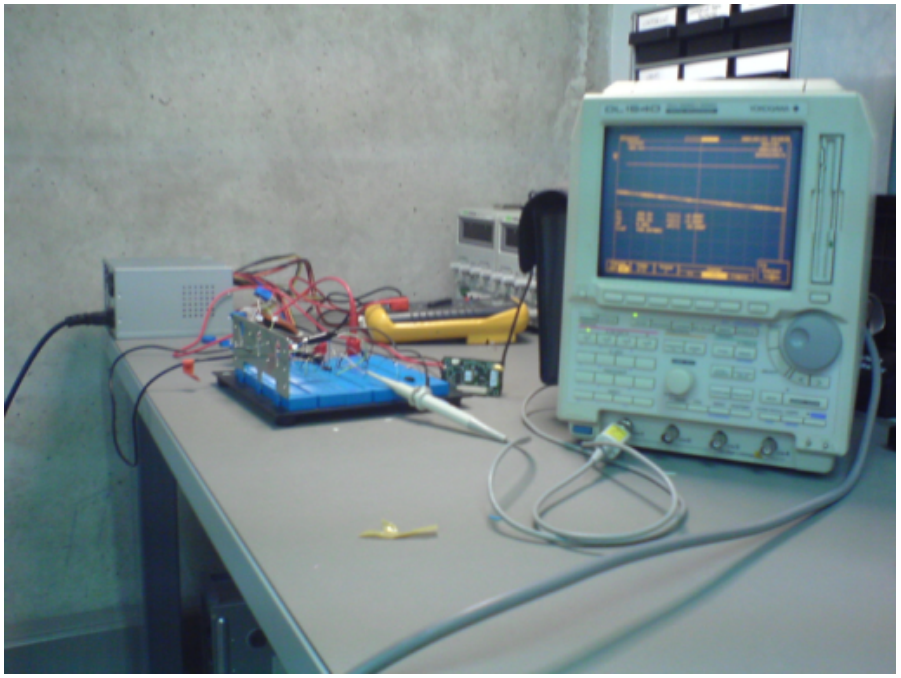


Figura 6.6: Se puede ver en el osciloscopio la pendiente negativa que proporciona el integrador a su salida

Capítulo 7

Problemas encontrados

Durante la realización de este proyecto se encontraron innumerables problemas para poder concluir su realización. A continuación se especifican algunos de ellos (los más significativos) para tratar de ilustrar la problemática asociada a la realización de un proyecto de este tipo e intentar ayudar a otros en posibles situaciones similares en las que se puedan encontrar.

Los problemas hallados durante la elaboración del proyecto fueron muy dispares, desde problemas con la adquisición de los propios dispositivos (Crossbow es una empresa norteamericana y el pedido se realizó a Estados Unidos) hasta problemas con las mediciones pasando, por supuesto, por problemas de software a la hora de implementar el TPC en el código de S-MAC.

7.1. Problemas software

Desde un primer momento la implementación de S-MAC utilizada (ver 4.2) produjo diversos problemas. Para empezar la documentación que acompañaba al protocolo no era demasiado extensa y se limitaba más a explicar el funcionamiento del protocolo [15] que a documentar el código entregado. Por otra parte, y en consecuencia de lo comentado, las aplicaciones proporcionadas eran escasas y de limitada funcionalidad (en la versión utilizada en el proyecto (ver 4.4) las aplicaciones que acompañan a S-MAC son cinco, dos que permiten comprobar la capa física para el envío y recepción, un *snooper*[*] y la aplicación *SMACTest* comentada en 4.5) por lo que los ejemplos para ayudar a su comprensión no abundaban. Esta conjunción de factores originó que se produjeran más problemas de los que en un principio se esperaban.

7.1.1. Aplicación de prueba

Los primeros problemas surgieron al intentar utilizar la aplicación de prueba *SMACTest*. Dicha aplicación, según la documentación que la acompaña, está configurada por defecto para ser instalada en tres nodos. El primero nodo (nodo 1) debe tener la dirección

[*]Un *snooper* es una aplicación que 'escucha' el medio y captura todos los mensajes que recibe (aunque no vayan destinados a él).

unicast '1' enviará paquetes al nodo 2 (igualmente la ID unicast debe estar definida) y, este último, enviará paquetes al tercer nodo que a su vez se los enviará al primero, cerrando así el anillo de transmisión. De este modo, al intentar programar los sensores para hacer la prueba, es necesario identificar cada uno de ellos asignándole su ID correspondiente pues, de otra forma, no existirá comunicación. Entre los mensajes unicast también se envían mensajes broadcast pero estos, lógicamente, no van dirigidos a ningún nodo específico.

En teoría existen dos maneras de asignar la ID unicast a los nodos, una es definiendo la macro *TST_UNICAST_ADDR* (ver el código del fichero *config.h* para más información) acompañada del ID del nodo y la otra es, al realizar la programación, incluyendo un punto seguido de la ID del nodo; de esta manera la llamada para invocar la programación de el nodo 1 sería algo como:

```
$:>MIB510=/dev/ttyS1 make install.1 mica2
```

El problema surgió porque la primera opción (asignando la ID a cada nodo a través de la macro) no funcionaba como es debido por lo que, por defecto, todos los nodos se programaban con el mismo ID (1). Una vez así los nodos no conseguían comunicarse entre sí puesto que todos mandaban mensajes con destino al nodo 2 pero, al no existir éste, solo se enviaban paquetes SYNC. Utilizando el segundo método todo funciona sin problemas pero hasta que se descubrió el fallo en la implementación, la realización de pruebas fue completamente imposible.

7.1.2. Potencia de transmisión por defecto

Otro obstáculo que se encontró en el camino hacia la implementación del TPC fue la modificación de la potencia de transmisión a través del método *setRFPower()*. En la documentación no figuraba ninguna referencia a la potencia de transmisión por defecto y, en la planificación de las pruebas, las estimaciones estaban hechas para una potencia de transmisión por defecto máxima (10dBm en el caso de los MICA2). La realidad era muy distinta y, en la implementación de la USC la potencia de transmisión por defecto era de 0dBm. El desconocimiento de ese dato hubiese manipulado totalmente los resultados en las medidas de consumo puesto que, al realizar las medidas sin TPC, los sensores estarían transmitiendo siempre a una potencia (0dBm) mientras que en la prueba con TPC los paquetes de datos variarían su potencia de transmisión mientras que los de control serían envíados a la potencia máxima (10dBm).

Para modificar el valor de la potencia por defecto es necesario adentrase en el códio de S-MAC y llegar hasta las capas más bajas de la implementación para modificar el registro encargado de almacenar el valor de la potencia de transmisión (*CC1K_PA_POW*). Para ello basta con editar el archivo *CC1000ControlM.nc* en la ruta *tos/platform/mica2/* para encontrar las siguientes líneas:

```
/*La potencia por defecto está a 0dBm pero se modifica para que,
  por defecto transmita a 10dBm (CC1K_PA_POW=0xFF)*/
    //gCurrentParameters[0xb] = ((8 << CC1K_PA_HIGHPOWER)
        | (0 << CC1K_PA_LOWPOWER));
        gCurrentParameters[0xb] = 0xFF;
    call HPLChipcon.write(CC1K_PA_POW, gCurrentParameters[0xb]);
```

El array *gCurrentParameters[]* guarda en su interior una serie de valores pero la posición 0xb está reservada para almacenar el valor del registro *CC1K_PA_POW*. Variando el valor de la posición 0xb del array se varía la potencia de transmisión puesto que es ese valor el que luego utiliza la interfaz de más bajo nivel *HPLChipcon*.

7.2. Problemas hardware

Al principio la idea para realizar las medidas era utilizar un osciloscopio que el departamento había comprado para las medidas. Dicho osciloscopio dispone de una interfaz RS-232 que, conectada al ordenador y utilizando el software proporcionado por el fabricante permite almacenar las muestras tomadas por el osciloscopio en un PC. Al probar el software y comenzar a utilizarlo se descubrió que el osciloscopio tenía un grave problema; a pesar de que toma 1MS/s (*Mega*[*] *Samples / second*; Muestras por segundo), a través de la interfaz serie solo puede enviar, como máximo, una cada 250 ms (4 muestras por segundo). Si tenemos en cuenta que los paquetes de datos de la aplicación de test ocupan 100 bytes (800 bits) y que el sensor transmite a 19,2 kbps:

$$t = \frac{800}{19,2 \cdot 10^3} = 0,041\overline{6} \cong 42ms \tag{7.1}$$

Si como se ha dicho, el osciloscopio envia muestras al PC, en el mejor de los casos cada 250 ms, las medidas tomadas mediante este método no reflejarán en absoluto el consumo real de los sensores pues, como salta a la vista, es posible que las medidas coincidan con el momento de una transmisión pero también es posible que no; eso sin tener en cuenta que los paquetes de control son mucho más pequeños (decenas de bits) y por supuesto, tampoco aparecerán contabilizados en las medidas.

7.2.1. Primera solución

Ante el problema surgido con el almacenamiento de muestras del osciloscopio se decidió recurrir al uso de otro osciloscopio disponible en el laboratorio. En este caso se trataba de un Yokogawa DL-1540 que es capaz de tomar hasta 20 millones de muestras por segundo (20 MS/s). Este osciloscopio dispone de una disquetera integrada que permite almacenar tanto muestras determinadas como capturas de cualquier medida realizada. Tras una pequeña investigación en el manual es fácil descubrir que este osciloscopio cumple sobradamente la frecuencia de muestreo, pero el problema surgía (como con el anterior) a la hora de almacenar las muestras.

Este modelo concreto solo soporta el almacenamiento de 10.000 muestras consecutivas por lo que, a la hora de realizar las pruebas, utilizando una frecuencia de muestreo de 100.000 muestras por segundo (100 kS/s), la mayor captura posible es de 100 ms. Lógicamente, una prueba de consumo de 100 ms no es válida para determinar el ahorro energético. Por lo cual esta solución se declaró como no válida y se optó, nuevamente, por buscar otra.

[*] Mega = 10^6

Figura 7.1: Imagen de un osciloscopio Yokogawa DL-1540

7.2.2. Segunda solución

Visto lo visto el principal problema era el almacenamiento de muestras para el posterior cálculo del consumo. Después de valorar diversas soluciones se decidió probar a usar la tarjeta de sonido de un ordenador como osciloscopio; de este modo se soluciona el problema del almacenamiento de muestras. Al fin y al cabo, se trata de un conversor A/D que toma muestras a través de la entrada de línea y de la del micrófono. En principio pareció ser la solución perfecta ya que la frecuencia de muestreo (44,1 kHz) era suficiente y el problema del almacenamiento dejaba de existir puesto que es posible importar la señal con cualquier programa de edición de sonido. Aunque estos programas no indiquen el nivel de voltaje, la amplitud que marquen puede ser fácilmente ponderada realizando previamente algunas pruebas a tensiones conocidas (si se introduce una tensión de un voltio es posible tomar una referencia de la amplitud que indica el programa y ponderar los resultados obtenidos). El primer paso fue desmontar un conector Jack de 3,5 mm estéreo como el de la figura 7.2.

Figura 7.2: Conector Jack de 3.5 mm estéreo

Una vez desmontado se procedió a conectar mediante pinzas el Jack al circuito, como se puede observar en la figura 7.3 se dispone de los dos canales y un tercer terminal para conectar a la masa. Se realizó un sencillo circuito de test para probar el nuevo "osciloscopio". Se conectó una resistencia de 1Ω en serie con un MICA2, estando éste alimentado por una fuente de tensión de 3V. De este modo midiendo en los bornes de la resistencia se obteiene la caída de tensión que indicará la corriente que circula por el circuito (y utilizando 6.1, 6.2 y 6.3 es posible obtener la energía consumida). Cuando el circuito estaba listo se enchufó el conector Jack a la entrada de línea de la tarjeta de sonido y, utilizando un programa editor de audio de libre distribución llamado Audacity dió comienzo la prueba.

Tras habilitar el micrófono desde el sistema operativo, desactivar el amplificador de 20dB que incorpora por defecto y comenzar la captura, se activó la fuente de alimentación.

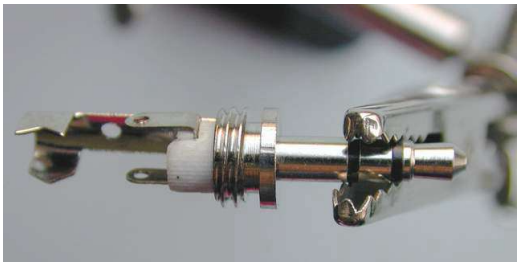


Figura 7.3: Conector Jack 3.5 mm desmontado

El programa comenzó a capturar una señal que, efectivamente variaba con los períodos de transmisión de paquetes de la aplicación de test (dicha aplicación utiliza los LEDs para indicar los periodos de actividad y la transmisión/recepción de paqutes). El problema venía en la forma que el programa tiene de representar la señal; ya que los resultados habían sido presumiblemente satisfactorios se investigó un poco más a través de internet. Existen bastantes programas distintos que sirven para simular un osciloscopio mediante la tarjeta de sonido. Después de probar algunos de ellos el más adecuado parecía *Soundcard Oscilloscope*, un software gratuito que permite su uso para fines educativos y no comerciales; su autor es el alemán Christian Zeitnitz y funciona perfectamente sobre la plataforma Windows funcionando además como generador de señales (aunque esta función resulta irrelevante en este caso).

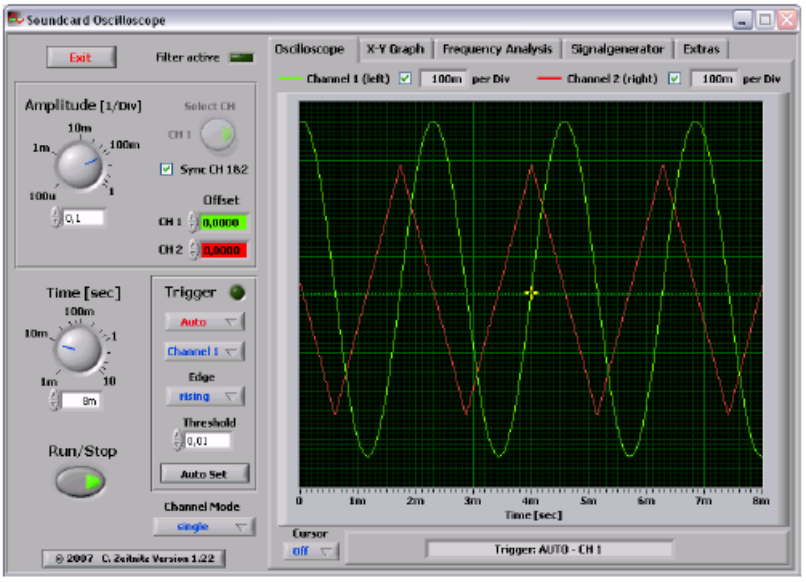


Figura 7.4: Captura del programa *Souncard Oscilloscope* funcionando sobre Windows XP

Se repitió la anterior prueba y esta vez la representación si era correcta, de hecho era aparentemente igual que la que se había obtenido con el anterior osciloscopio (ver figura 7.1). Tras observar durante un tiempo la representación se pudo apreciar que la señal, al

estabilizarse se centraba en cero, es decir, los períodos en los que el voltaje era constante la caída de tensión que representaba el programa era 0V. Tras consultar diversas fuentes se descubrió que las tarjetas de sonido incluyen un condensador a su entrada que se utiliza para filtrar la componente continua de las señales de entrada por lo que, una vez más, no pudimos utilizar esta solución.

7.2.3. Tercera y última solución

Tras el último fracaso la opción más sensata era probar una nueva solución; desempolvando los apuntes de Electrónica Analógica apareció un circuito que podía realizar, de manera autónoma, el proceso de almacenamiento de las distintas muestras, además, con mucha más precisión; se trata del circuito integrador.

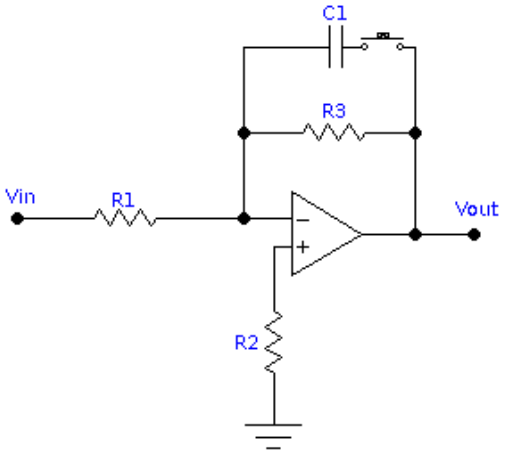


Figura 7.5: Circuito integrador realizado con un amplificador operacional

Dicho circuito saca a su salida la integral de la señal que recibe a su entrada multiplicada por un factor dependiente de los valores de R1 y C1; la tensión de salida viene dada por la siguiente expresión:

$$V_{OUT} = -\frac{1}{R1 \cdot C1} \int_{t1}^{t2} V_{IN} dt \tag{7.2}$$

De manera que si se conecta a la entrada de dicho circuito la caída de tensión en una resistencia en serie con el sensor, y a la salida un osciloscopio que mida el voltaje de salida (Vout) se obtiene, en dicho osciloscopio la suma de todos los valores a la entrada del circuito. Si la resistencia es lo suficientemente baja ($\preceq 10\Omega$) no influirá en el consumo por lo que solo quedará multiplicar la tensión de salida (V_{OUT}) por el factor mencionado ($R1 \cdot C1$), dividirlo por la resistencia sobre la que se mide la caída y multiplicarlo por la tensión de alimentación:

$$E = \frac{V_{OUT} \cdot R1 \cdot C1}{R_{SHUNT}} \cdot V_{CC} \tag{7.3}$$

Para empezar hubo que conseguir todo el material necesario:

- Una placa board.
- Un amplificador operacional OPA602AP.
- Dos resistencias de 3MΩ (R1 = R2 hace que disminuya el error de offset).
- Un condensador de 10μF.
- Una resistencia (R3) de 220Ω.
- Una resistencia de 10Ω.
- Un interruptor que cortocircuita el condensador y permite su descarga a través de la resistencia R3.
- Tres fuentes de alimentación ($\pm$12V para alimentar el operacional y 3,3V para el MICA2).

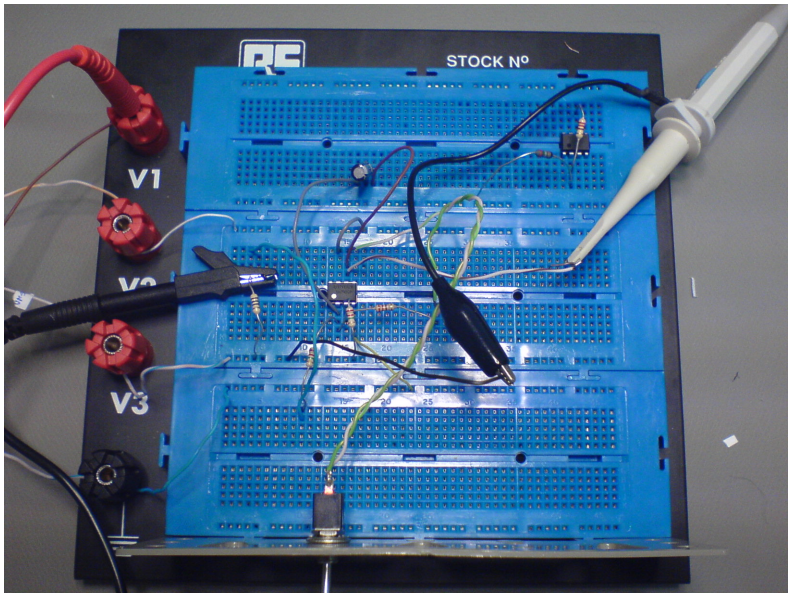


Figura 7.6: Circuito integrador montado en el laboratorio

Tras un buen rato montando y probando el circuito sobre la placa board se terminó el diseño y quedó como muestra la figura 7.6. Se realizaron algunas pruebas con resistencias constantes para comprobar como se comportaba, cuanto tiempo tardaba en entrar en saturación, qué tensión alcanzaba en dicha zona... Tras unas cuantas pruebas más, el circuito estaba listo para realizar su tarea, solo había un pequeño inconveniente, se estaban utilizando tres fuentes de alimentación distintas lo cual es muy aparatoso y, además, una de ellas era necesaria en el laboratorio ya que otro grupo debía usarla.

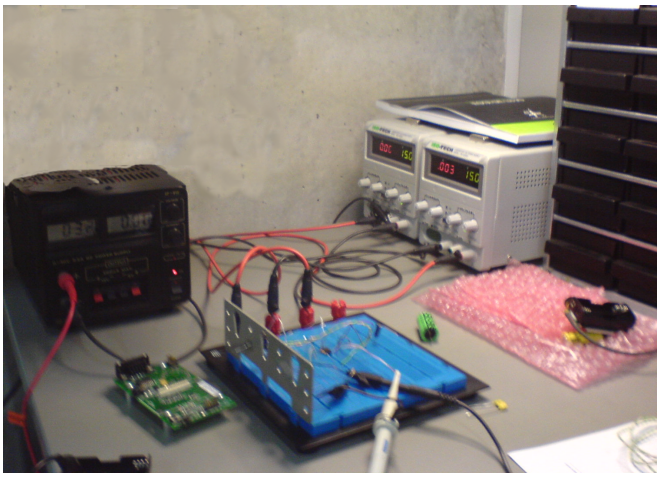


Figura 7.7: Montaje inicial realizado en el laboratorio con tres fuentes de alimentación

7.2.4. Fuentes de alimentación

Como ya se ha comentado, disponer de tres fuentes de alimentación es demasiado aparatoso, eso sin contar el hecho de que, realmente, solo era posible utilizar dos de ellas. Pensando las diversas opciones que quedanban se decidió utilizar una fuente de alimentación de un PC. Tras conseguir el esquema de voltaje del conector de una fuente *Advanced Technology Extended* (ATX) se comprobó que, si era posible utilizarla, era perfecta, ya que proporcionaba todos los niveles de tensión que el circuito de medidas requería ($\pm$12V y 3,3V).

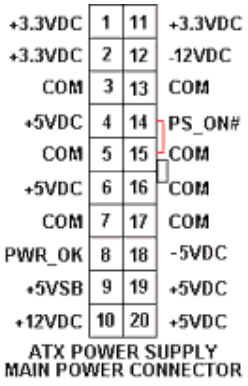


Figura 7.8: En rojo se pueden ver los dos pines que es necesario conectar para encender la fuente.

Las fuentes de alimentación ATX en principio, no pueden conectarse solas, debe ser la placa base de un PC la que las 'encienda'. La placa base, cuando se presiona el pulsador de encendido dispara un tiristor que queda conduciendo y "puentea" dos terminales de la fuente produciendo el encendido de la misma (ver en figura 7.8 el puenteado rojo). Sabiendo esto, para poder encender la fuente, se soldó un interruptor a los dos pines de la fuente que había que unir para poder apagar y encender la susodicha cuando fuese necesario.

Además, se utilizaron unas regletas para poder pinchar ahí los cables y extraer las distintas tensiones necesarias para alimentar el circuito. Tras asegurar un poco las conexiones, etiquetar adecuadamente todos los cables y realizar algunas medidas de prueba, la fuente estaba totalmente lista para su uso con los MICA2 y el circuito de test. f

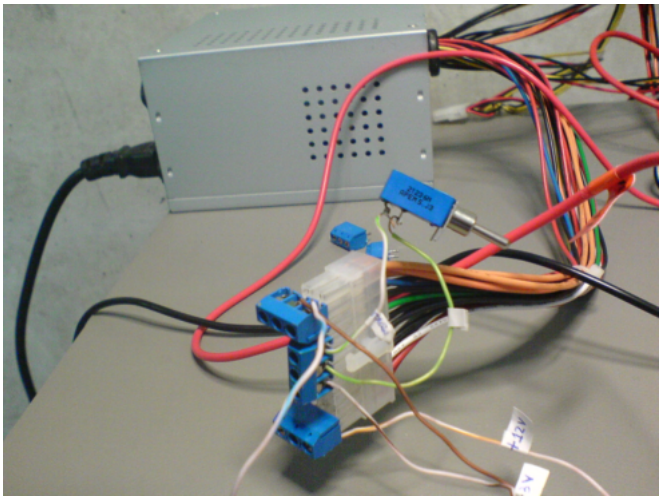


Figura 7.9: Conector ATX con las modificaciones realizadas para usarlo en el circuito de medidas

Capítulo 8

Conclusiones y líneas futuras

Al observar los resultados obtenidos en las simulaciones y pruebas, es posible verificar lo que ya se suponía gracias al estudio teórico [1] realizado previamente al respecto. Las irregularidades mostradas en la figura 6.5 hacen pensar que es posible conseguir ahorros de hasta el 12 % por lo que, la cuestión que ahora se plantea es, ¿qué factores hacen que las medidas se alteren tanto de una prueba a otra?. Esas variaciones tan bruscas no pueden obviarse y, a la vista de los resultados la opción más coherente no es otra que indicar que los resultados no son concluyentes.

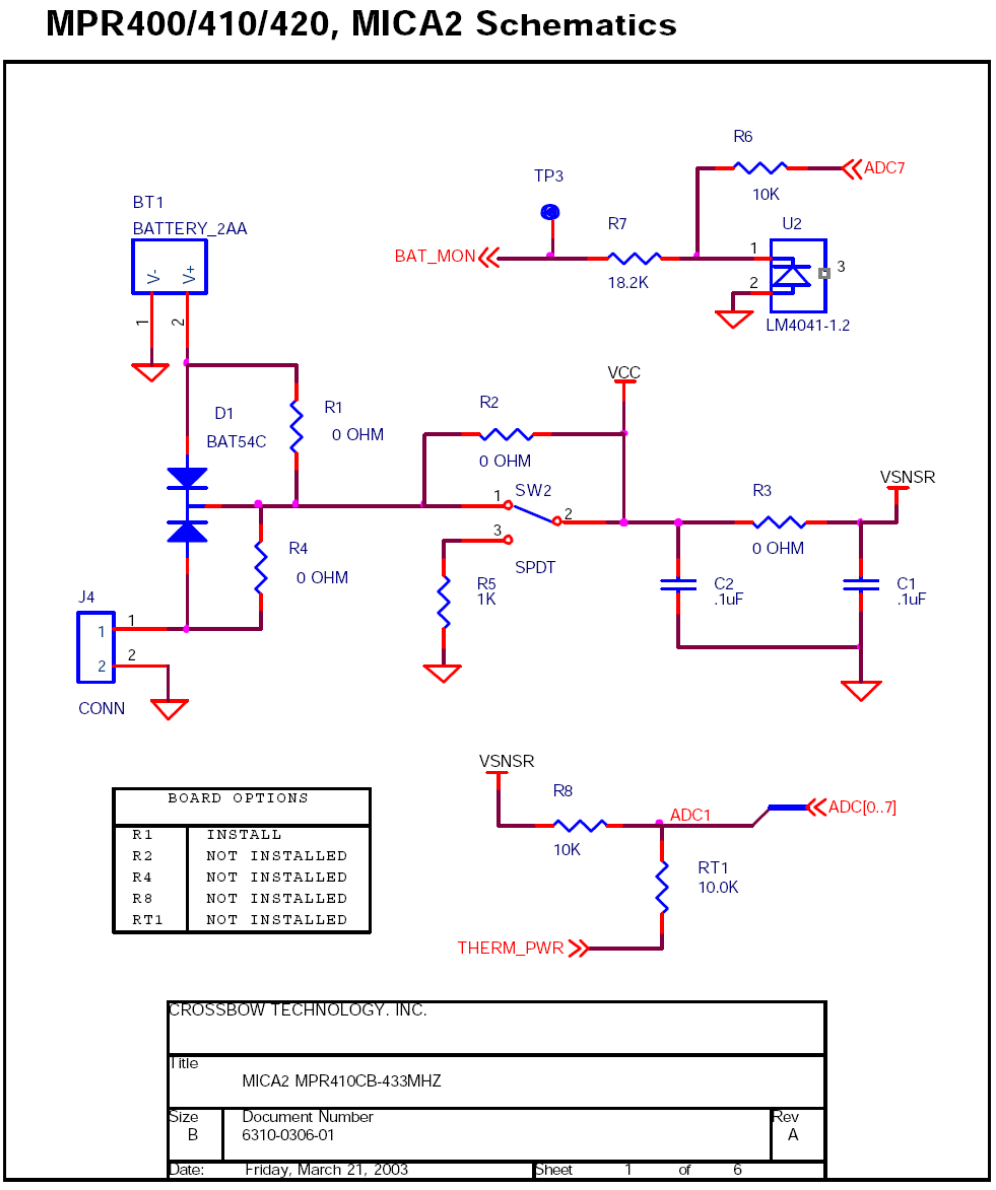
Por otra parte y pensando en futuras líneas de investigación relacionadas con el proyecto, se plantea la posibilidad de implementar el mecanismo de control de potencia con otros protocolos cuyo rendimiento vaya a ser más notable como por ejemplo L-MAC [1] o incluso se podría mejorar la implementación existente del control de potencia en S-MAC para hacerlo más preciso.

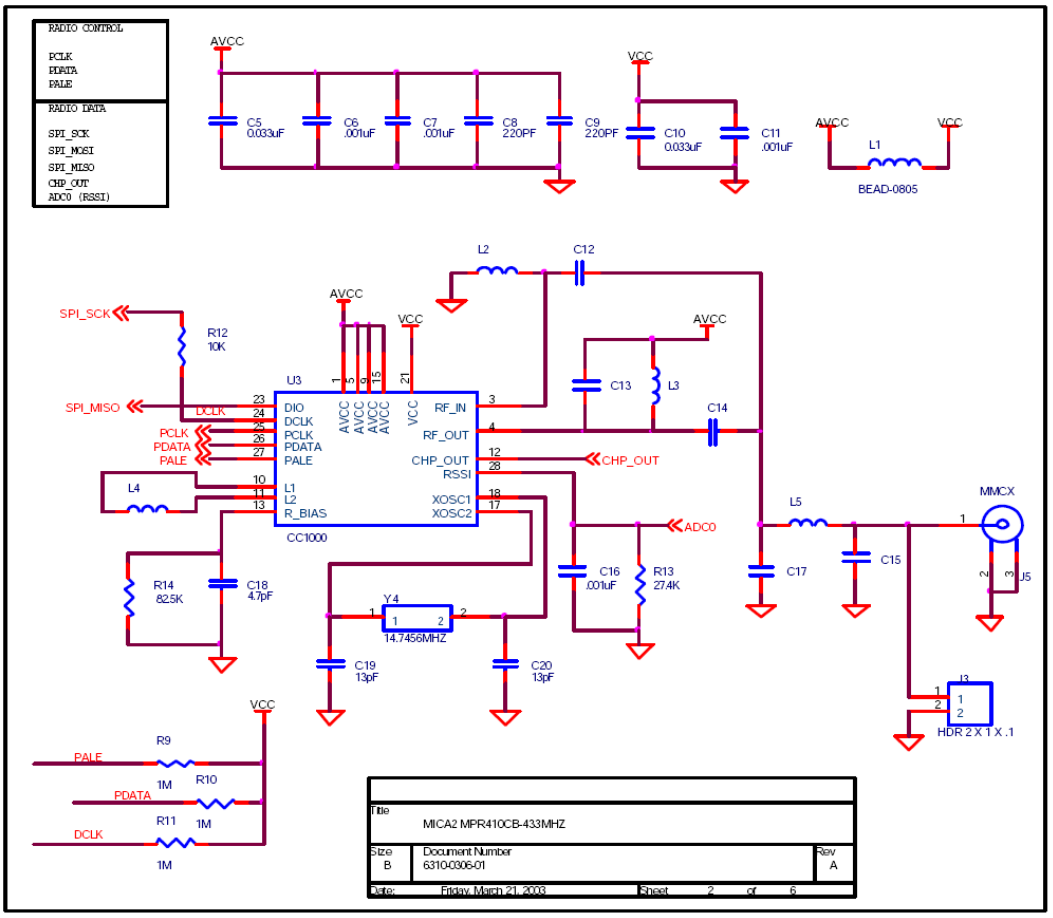
Por otro lado, el circuito de medida puede ser mejorado, realizando un diseño un poco más complejo es posible integrar un temporizador que controle por sí solo la duración de las pruebas. Por otra parte continuando en la misma línea, el circuito integrado puede intentar reducirse para integrarlo en el lugar donde, en la actualidad, se colocan las baterías en los sensores; de este modo la medición se realizará de manera integrada sin que el añadido de la placa aumente en el tamaño final. Además sería posible la integración en un solo conector para toda la alimentación del circuito en lugar de utilizar regletas como se hace en la actualidad.

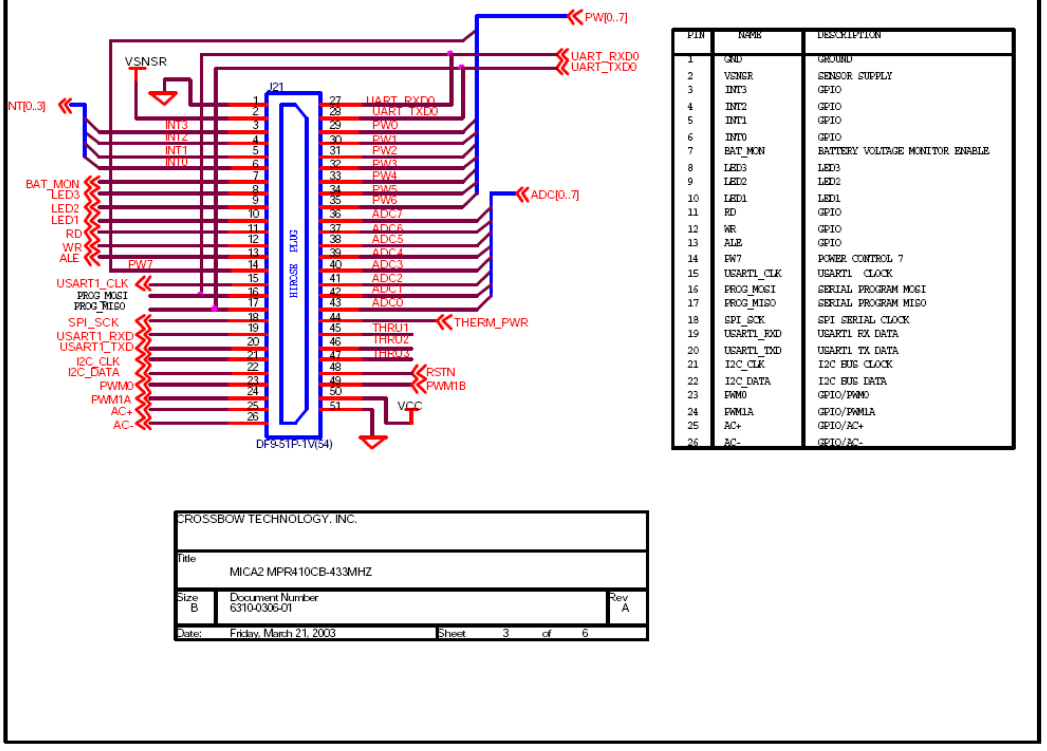
Finalmente reiterar el hecho de que los resultados obtenidos en estas primeras pruebas no han resultado concluyentes por lo que, antes de seguir con cualquier investigación es recomendable intentar encontrar alguna explicación coherente que justifique los datos recogidos.

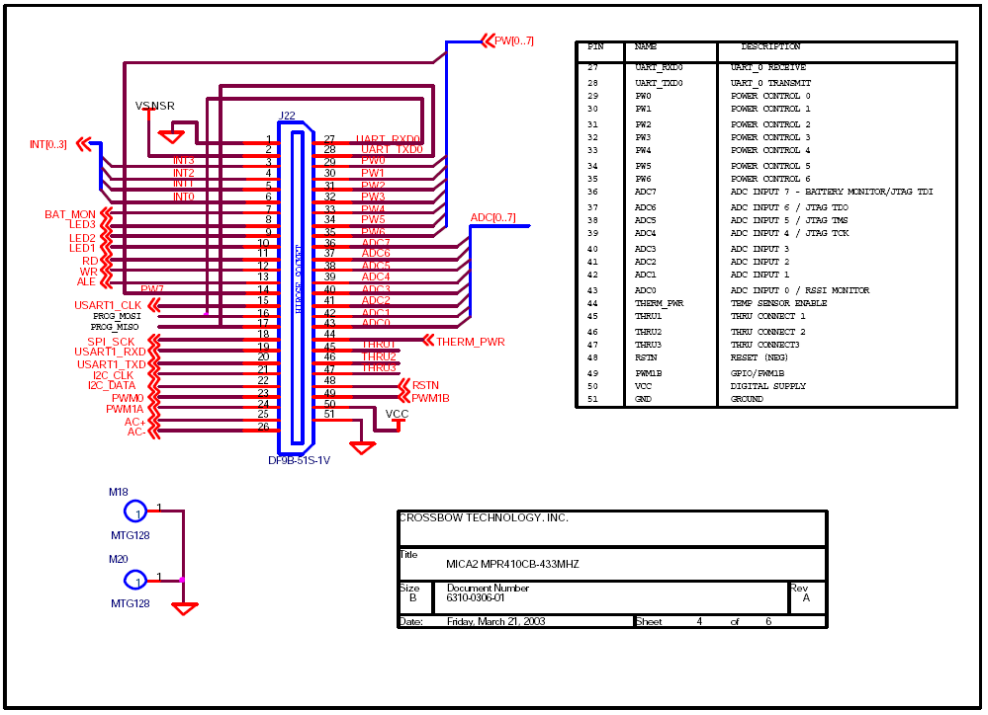
Apéndice A

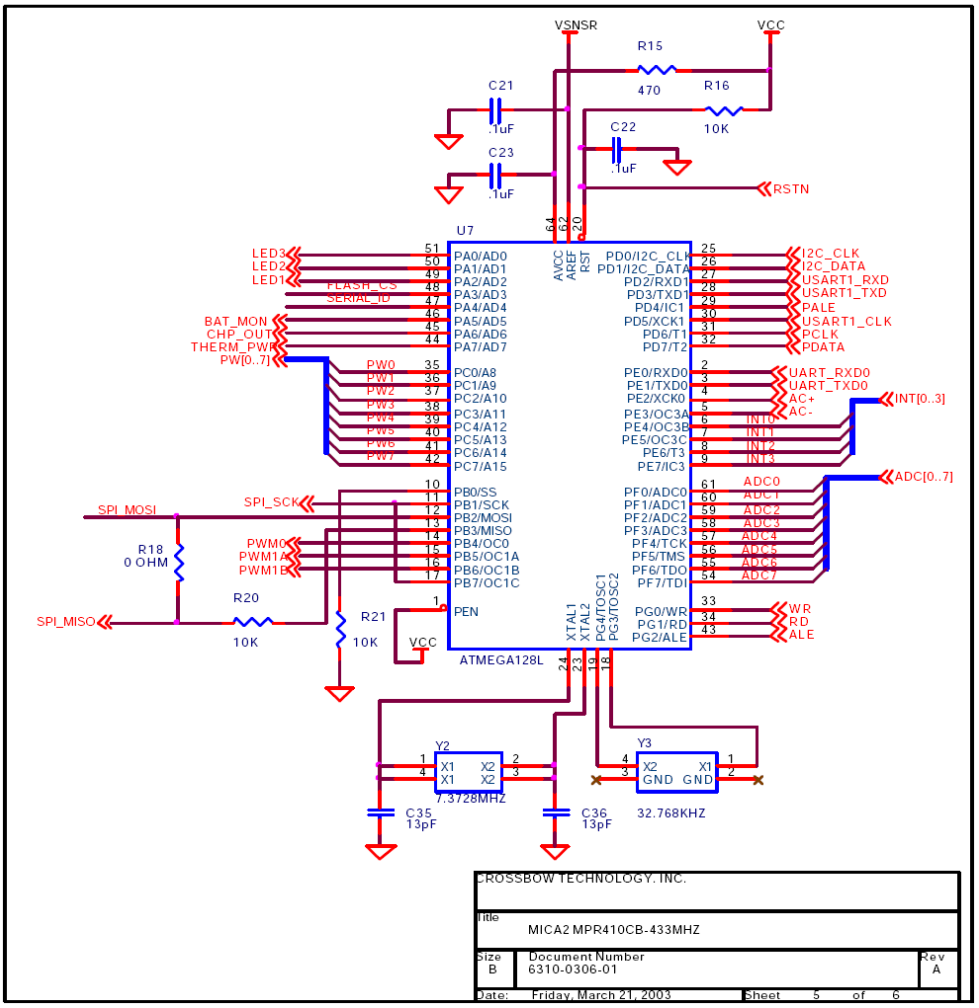
Esquemas y circuitos de los MICA2

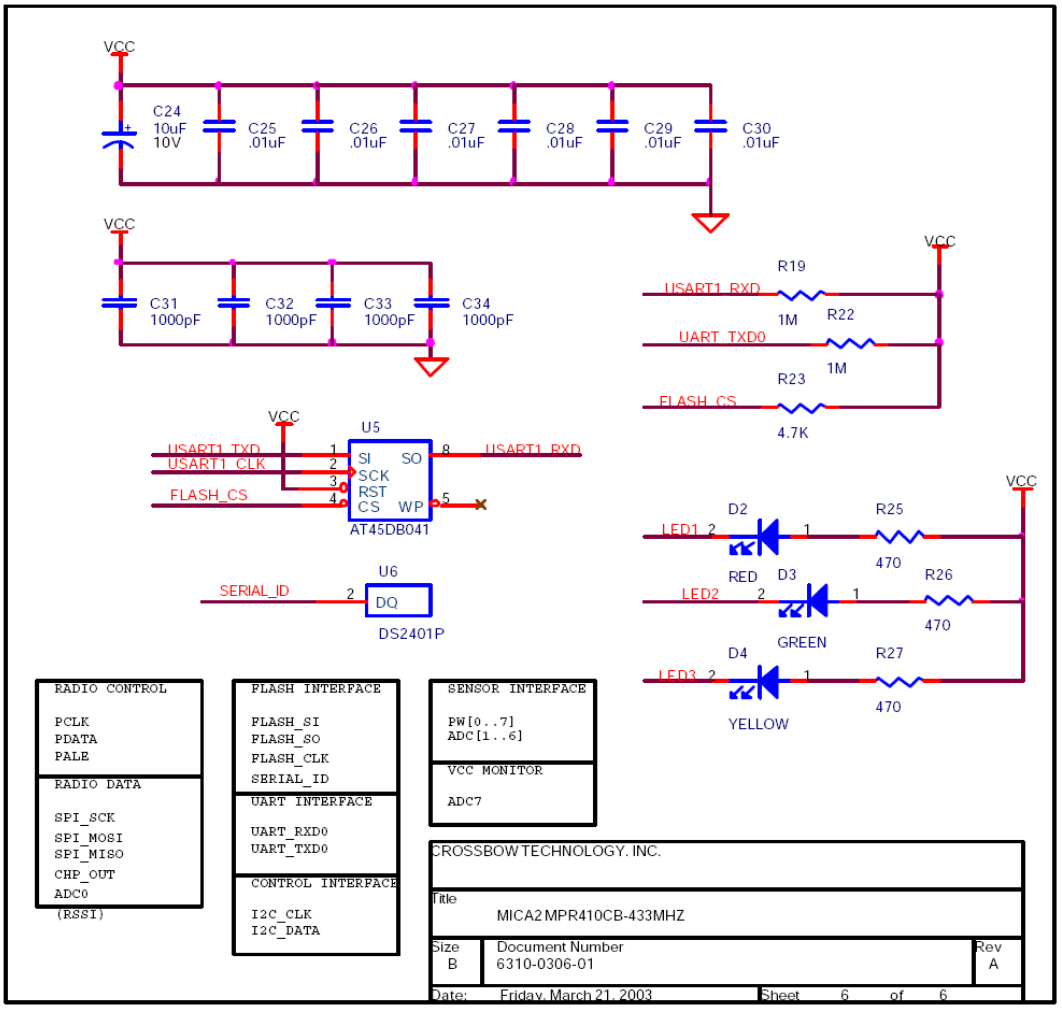



RADIO CONTROL
PCLK
PDATA
PALE
RADIO DATA
SPI_SCK
SPI_MOSI
SPI_MISO
CHP_OUT
ADC0 (RSSI)
AVCC
VCC
C5 0.033uF
C6 .001uF
C7 .001uF
C8 220PF
C9 220PF
C10 0.033uF
C11 .001uF
L1
BEAD-0805
L2
C12
R12 10K
U3
DIO
DCLK
PCLK
PDATA
PALE
RF_IN
RF_OUT
CHP_OUT
RSSI
XOSC1
XOSC2
R_BIAS
CC1000
C13
L3
C14
L4
L5
C15
C16 .001uF
R13 27.4K
C17
C18 4.7pF
R14 82.5K
Y4
14.7456MHZ
C19 13pF
C20 13pF
MMCX
J5
J3
HDR 2 X 1 X .1
R9 1M
R10 1M
R11 1M
Title
MICA2 MPR410CB-433MHZ
Size B
Document Number 6310-0306-01
Rev A
Date: Friday, March 21, 2003
Sheet 2 of 6


PW[0..7]
UART_RXD0
UART_TXD0
VSNSR
INT[0..3]
J21
HIROSE FLEX
DF9-51P-1V(54)
ADC[0..7]
THERM_PWR
PSTN
PWM1B
BAT_MON
LED3
LED2
LED1
RD
WR
ALE
USART1_CLK
PROG_MOSI
PROG_MISO
SPI_SCK
USART1_RXD
USART1_TXD
I2C_CLK
I2C_DATA
PWM0
PWM1A
AC+
AC-
PIN NAME DESCRIPTION
1 GND GROUND
2 VSNSR SENSOR SUPPLY
3 INT3 GPIO
4 INT2 GPIO
5 INT1 GPIO
6 INT0 GPIO
7 BAT_MON BATTERY VOLTAGE MONITOR ENABLE
8 LED3 LED3
9 LED2 LED2
10 LED1 LED1
11 RD GPIO
12 WR GPIO
13 ALE GPIO
14 PW7 POWER CONTROL 7
15 USART1_CLK USART1 CLOCK
16 PROG_MOSI SERIAL PROGRAM MOSI
17 PROG_MISO SERIAL PROGRAM MISO
18 SPI_SCK SPI SERIAL CLOCK
19 USART1_RXD USART1 RX DATA
20 USART1_TXD USART1 TX DATA
21 I2C_CLK I2C BUS CLOCK
22 I2C_DATA I2C BUS DATA
23 PWM0 GPIO/PWM0
24 PWM1A GPIO/PWM1A
25 AC+ GPIO/AC+
26 AC- GPIO/AC-
CROSSBOW TECHNOLOGY, INC.
Title
MICA2 MPR410CB-433MHZ
Size B
Document Number 6310-0306-01
Rev A
Date: Friday, March 21, 2003
Sheet 3 of 6


PW[0..7]
VSNSR
J22
INT[0..3]
INT3
INT2
INT1
INT0
BAT_MON
LED3
LED2
LED1
RD
WR
ALE
PW7
USART1_CLK
PROG_MOSI
PROG_MISO
SPI_SCK
USART1_RXD
USART1_TXD
I2C_CLK
I2C_DATA
PWM0
PWM1A
AC+
AC-
UART_RXD0
UART_TXD0
PW0
PW1
PW2
PW3
PW4
PW5
PW6
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
THRU1
THRU2
THRU3
RSTN
PWM1B
VCC
ADC[0..7]
THERM_PWR
DF9B-51S-1V
M18
MTG128
M20
MTG128
PIN
NAME
DESCRIPTION
27 UART_RXD0 UART_0 RECEIVE
28 UART_TXD0 UART_0 TRANSMIT
29 PW0 POWER CONTROL 0
30 PW1 POWER CONTROL 1
31 PW2 POWER CONTROL 2
32 PW3 POWER CONTROL 3
33 PW4 POWER CONTROL 4
34 PW5 POWER CONTROL 5
35 PW6 POWER CONTROL 6
36 ADC7 ADC INPUT 7 - BATTERY MONITOR/JTAG TDI
37 ADC6 ADC INPUT 6 / JTAG TDO
38 ADC5 ADC INPUT 5 / JTAG TMS
39 ADC4 ADC INPUT 4 / JTAG TCK
40 ADC3 ADC INPUT 3
41 ADC2 ADC INPUT 2
42 ADC1 ADC INPUT 1
43 ADC0 ADC INPUT 0 / RSSI MONITOR
44 THERM_PWR TEMP SENSOR ENABLE
45 THRU1 THRU CONNECT 1
46 THRU2 THRU CONNECT 2
47 THRU3 THRU CONNECT3
48 RSTN RESET (NEG)
49 PWM1B GPIO/PWM1B
50 VCC DIGITAL SUPPLY
51 GND GROUND
CROSSBOW TECHNOLOGY, INC.
Title
MICA2 MPR410CB-433MHZ
Size B
Document Number 6310-0306-01
Rev A
Date: Friday, March 21, 2003
Sheet 4 of 6


VSNSR
VCC
R15
470
R16
10K
C21
.1uF
C22
.1uF
C23
.1uF
RSTN
U7
ATMEGA128L
LED3
LED2
LED1
FLASH_CS
SERIAL_ID
BAT_MON
CHP_OUT
THERM_PWR
PW[0..7]
I2C_CLK
I2C_DATA
USART1_RXD
USART1_TXD
PALE
USART1_CLK
PCLK
PDATA
UART_RXD0
UART_TXD0
AC+
AC-
INT[0..3]
ADC[0..7]
SPI_SCK
SPI_MOSI
SPI_MISO
R18
0 OHM
R20
10K
R21
10K
PWM0
PWM1A
PWM1B
WR
RD
ALE
Y2
7.3728MHZ
Y3
32.768KHZ
C35
13pF
C36
13pF
CROSSBOW TECHNOLOGY, INC.
Title
MICA2 MPR410CB-433MHZ
Size B
Document Number 6310-0306-01
Rev A
Date: Friday, March 21, 2003
Sheet 5 of 6


VCC
C24
10uF
10V
C25
.01uF
C26
.01uF
C27
.01uF
C28
.01uF
C29
.01uF
C30
.01uF
VCC
C31
1000pF
C32
1000pF
C33
1000pF
C34
1000pF
VCC
R19
USART1_RXD
1M
R22
UART_TXD0
1M
R23
FLASH_CS
4.7K
VCC
U5
USART1_TXD
USART1_CLK
FLASH_CS
SI
SCK
RST
CS
SO
WP
USART1_RXD
AT45DB041
U6
SERIAL_ID
DQ
DS2401P
VCC
D2
LED1
R25
470
RED
D3
LED2
R26
470
GREEN
D4
LED3
R27
470
YELLOW
RADIO CONTROL
PCLK
PDATA
PALE
RADIO DATA
SPI_SCK
SPI_MOSI
SPI_MISO
CHP_OUT
ADC0
(RSSI)
FLASH INTERFACE
FLASH_SI
FLASH_SO
FLASH_CLK
SERIAL_ID
UART INTERFACE
UART_RXD0
UART_TXD0
CONTROL INTERFACE
I2C_CLK
I2C_DATA
SENSOR INTERFACE
PW[0..7]
ADC[1..6]
VCC MONITOR
ADC7
CROSSBOW TECHNOLOGY, INC.
Title
MICA2 MPR410CB-433MHZ
Size
B
Document Number
6310-0306-01
Rev
A
Date:
Friday, March 21, 2003
Sheet
6
of
6

Acrónimos y abreviaturas

MAC *Medium Access Control*

S-MAC *Sensor-Medium Access Control*

T-MAC *Time-Medium Access Control*

PDA *Personal Device Assistant*

PC *Personal Computer*

WIFI *Wireless Fidelity*

IEEE *International Electrical and Electronical Engineers*

WSN *Wireless Sensors Network*

MHz *Mega Hertzios*

RSSI *Receive Signal Strength Indicator*

A/D *Analógico / Digital*

ADC *Analog to Digital Converter*

S/N *Signal / Noise*

ISP *In System Processor*

RF *Radio Frecuencia*

TPC *Transmission Power Control*

RTS *Request To Send*

CTS *Clear To Send*

ACK *ACKnowledgement*

ATX *Advanced Technology Extended*

ISI *Information Sciences Institute*

USC *University of Southern California*

GPLv2 *General Public License version 2.0*

UART *Universal Asynchronous Receiver-Transmitter*

PCB *Printed Circuit Board*

Bibliografía

[1] Performance Evaluation of MAC Transmission Power Control in Wireless Sensor Networks. *Javier Vales Alonso, Esteban Egea-López, Alejandro Martínez-Sala, Pablo Pavón-Mariño, M. Victoria Bueno-Delgado, Joan García-Haro.*

[2] Wireless Sensor Networks. "Smart Environments: Technologies, Protocols, and Applications" *F. L. LEWIS, Ed. D.J. Cook and S.K. Das, John Wiley*, New York, 2004. Web: http://arri.uta.edu/acs/networks/WirelessSensorNetChap04.pdf

[3] Sensor Networks: An Overview. *Archana Bharathidasan, Vijay Anand Sai Ponduru* Web: http://wwwcsif.cs.ucdavis.edu/ bharathi/sensor/survey.pdf

[4] Networking Issues in Wireless Sensor Networks *Deepak Ganesan, Alberto Cerpa*. Web: http://www.isi.edu/ weiye/pub/jpdc.pdf

[5] TinyOS. An Operative System for Wireless Sensor Networks. *Mark Weiss*. University of Nebraska, Lincoln. 2003.
Web: http://csce.unl.edu/ witty/f2004/csce489/WSN-20031201.pdf

[6] Página web oficial del proyecto TinyOS
Web: http://www.tinyos.net/

[7] Wikipedia, la enciclopedia libre
Web: http://es.wikipedia.org/

[8] Página web de Crossbow Technology
Web: http://www.xbow.com/

[9] Página web de Moteiv
Web: http://www.moteiv.com/

[10] Página web de Shockfish
Web: http://www.shockfish.com/

[11] Página web de Chipcon
http://www.chipcon.com/

[12] nesC: A Programming Language for Deeply Networked Systems
Web: http://nescc.sourceforge.net/

[13] Wei Ye's Home Page *Wei Ye*.
Web: http://www.isi.edu/ weiye/

[14] S-MAC Software: Information and Source Code *Wei Ye, Padma Haldar, Yuan Li, Honghui Chen, Vijaykumar Kakadia*.
Web: http://www.isi.edu/ilense/software/smac/index.html

[15] An Energy-Efficient MAC Protocol for Wireless Sensor Networks *Wei Ye, John Heidemann, Deborah Estrin*.
Web: http://www.isi.edu/ weiye/pub/smac_infocom.pdf

[16] Energy Aware Routing for Low Energy Ad Hoc Sensor Networks *R. C. Shah, J. Rabaey*.
IEEE Wireless Communications and Networking Conference (WCNC), March 17-21, 2002, Orlando, FL.

[17] Trabajo de Ampliación de Redes *Jesús Serna Sanchís* .
http://www.uv.es/∼montanan/ampliacion/trabajos/Redes %20de %20Sensores.pdf